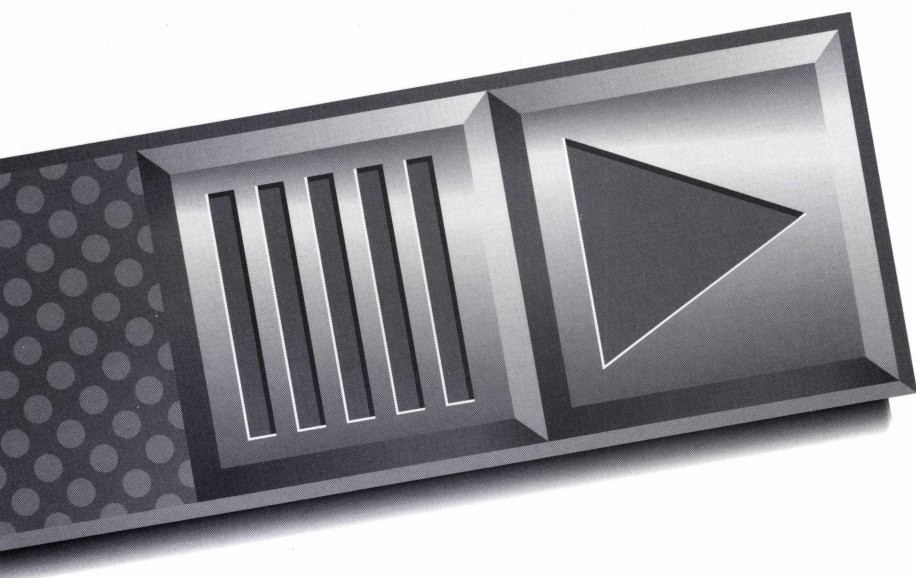
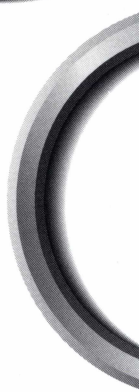
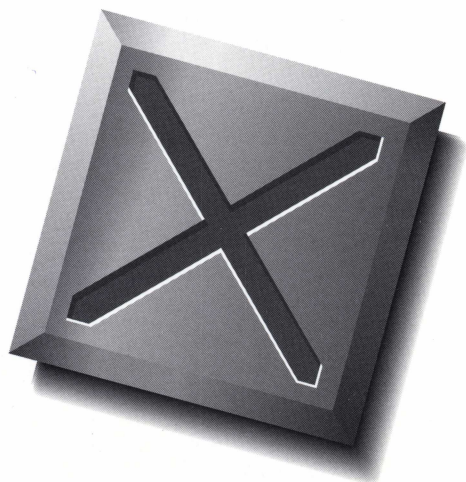


XVT



Introduction to *XVT* Development for C++

Introduction to XVT Development for C++

Presented by
Steve Hansen
August 16-17, 1993

XVT Software Inc.
4900 Pearl East Circle
Boulder, CO 80301 USA
(303)443-4223

Copyright © 1993. All Rights Reserved

[illegible][illegible]

10

1952

and

774

8796

1910

Section 1

Course Introduction

 Introduction to XVT-Design++ Development


Slide 1

Our First XVT-Design++ Application

❖ A small example application:

- ♦ Layout GUI objects
- ♦ Set connections
- ♦ Write action code
- ♦ Generate files
- ♦ Build application

❖ Larger applications can be just as easy to implement

 Introduction to XVT-Design++ Development

Slide 2

How the Course is Structured

- ❖ Section 1 - Course Introduction and GUI Concepts
- ❖ Section 2 - XVT-Design++ Overview
- ❖ Section 3 - Planning an XVT-Design++ Application
- ❖ Section 4 - Building an XVT-Design++ Application
- ❖ Section 5 - Building an XVT-Design++ Application
- ❖ Section 6 - Completing an XVT-Design++ Application

Introduction to XVT-Design++ Development

Slide 3

What This Section Covers

- ❖ XVT and Course Introduction
- ❖ Review of GUI Concepts
 - ♦ Event-Driven Programming
 - ♦ Approaches to Portability
 - ♦ Native Window System Differences
- ❖ Terminology and Typographical Conventions
- ❖ Course Example Application

Introduction to XVT-Design++ Development

Slide 4

Who and What is XVT?

- ❖ eXtensible Virtual Toolkit
- ❖ XVT provides portable, easy to use GUI development environments
- ❖ C and C++ based products are available for 7 different operating systems and 30 hardware platforms
- ❖ XVT was founded by Marc Rochkind in 1988
- ❖ XVT is headquartered in Boulder CO and currently has a full-time staff of over 100

Introduction to XVT-Design++ Development

Slide 5

Course Goals

- ❖ Oriented toward introductory process and techniques (not an advanced class)
- ❖ Learn the basics of portable GUI programming in C++
- ❖ Gain familiarity with XVT-Design++ and XVT++
 - ♦ Practical approaches to XVT programming
 - ♦ Not intended to cover all aspects of XVT products
 - ♦ Functionality that's used most often
- ❖ Teach you to help yourselves - through documentation and example code
- ❖ Cross-platform understanding and appreciation

Introduction to XVT-Design++ Development

Slide 6

Course Prerequisites

- ❖ Intermediate C++ programming expertise (advanced preferred - not required)
- ❖ Good familiarity with use of target GUI platform(s)
- ❖ No GUI programming experience required (but certainly helpful)

Introduction to XVT-Design++ Development

Slide 7

Typical Daily Course Schedule

- ❖ Two sessions per day, from:
 - ♦ 8:30 to 12:30
 - ♦ 1:30 to 5:30
- ❖ There will be 1 or 2 breaks during each session

Introduction to XVT-Design++ Development

Slide 8



Course Materials

❖ **Provided**

- ♦ "Introduction to XVT-Design++ Development" Course Notes
- ♦ Student Background Information Form
- ♦ Course Evaluation Form

❖ **Recommended**

- ♦ XVT-Design++ Volume 1 - *Using XVT-Design++*
- ♦ XVT-Design++ Volume 2 - *XVT++ 2.0 Class Library Reference*
- ♦ XVT-Design++ *Quick Reference Guide*



Introduction to XVT-Design++ Development

Slide 9



Review of GUI Concepts

- ❖ **Line Output-Oriented Applications**
- ❖ **GUI (Event-Driven) Programming**
- ❖ **Portable GUI Applications**
- ❖ **Differences between Native GUI Systems**



Introduction to XVT-Design++ Development

Slide 10

Line Output-Oriented Applications

- ❖ **Application logic, not the user, is in control**
 - ♦ Inflexible interaction - user must follow program
 - ♦ Control flow clear from examining source code
- ❖ **I/O is a stream of characters**
 - ♦ Only one type of input or output (characters)
 - ♦ Limited to prompt-response style (screen) interfaces
 - ♦ Ability to pipe input or output (ie easier testing)
- ❖ **Output model: row-column character grid**
 - ♦ Effectively an extension of the typewriter

Introduction to XVT-Design++ Development

Slide 11

Line Output-Oriented Applications (cont.)

- ❖ **Interface code often intertwined with application code**
 - ♦ Simple I/O code often scattered throughout program
 - ♦ Output lines may scroll off screen
- ❖ **Can not produce accurate graphical images**
- ❖ **Limited character attributes: character, color, highlighting/blinking**

Introduction to XVT-Design++ Development

Slide 12

The GUI Desktop Metaphor

❖ The Virtual Desktop

- ♦ Screen == Desktop
- ♦ Application programs == Work tools
- ♦ Work organized via files and folders/directories

❖ Working with multiple applications

- ♦ Overlapping applications
- ♦ Resizing of application windows (each with multiple views of work)
- ♦ GUI object drag and drop actions

 Introduction to XVT-Design++ Development

Slide 13

GUI Applications

❖ User, *not* application logic, is in control

- ♦ Application structured to respond to user interaction (events)
- ♦ Application control flow depends on user interaction

❖ Several types of input events to handle

- ♦ Keyboard, mouse, scrolling, window system, etc.
- ♦ User can directly manipulate application components

❖ All output is rendered graphically

- ♦ Text as well as graphical shapes
- ♦ Pixel-based direct access of screen locations
- ♦ High-quality graphical symbols and images possible
- ♦ Visual interface usually represents a virtual desktop

 Introduction to XVT-Design++ Development

Slide 14

GUI Applications (cont.)

- ❖ Interface code separate from application code
- ❖ Desktop metaphor requires ability to redraw information at any time
 - ♦ Stacking order changes
 - ♦ Window size or location changes
 - ♦ Menus and dialogs may invalidate regions of windows
- ❖ GUI layout information can be separated from source code

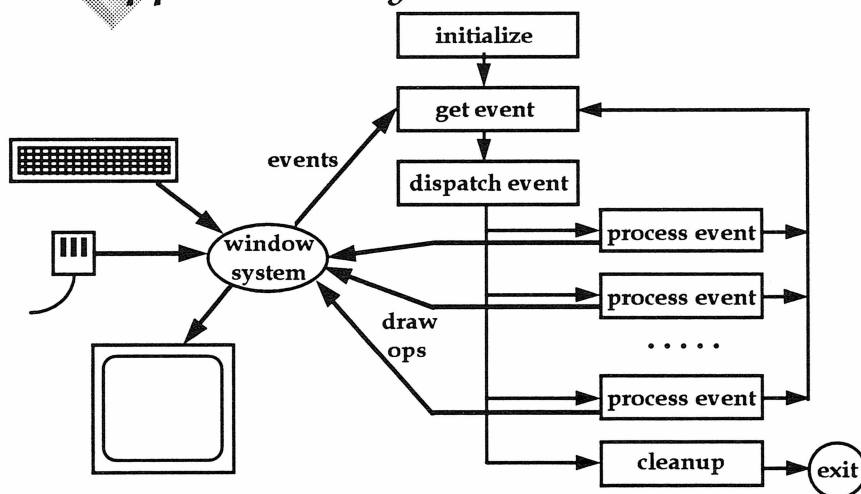
The GUI is Event-Driven

- ❖ Events *drive* GUI applications
 - ♦ Events are generated by the user, the application, or the system
- ❖ The application is structured to respond to events
 - ♦ Classes define member functions for the events they respond to
 - ♦ The XVT++ Class Library provides virtual member functions
 - ♦ You will write application-specific member functions

GUI Event-Driven Program Structure

- ❖ All input events come in through a central event fetching and dispatching loop
- ❖ Events can occur in any order at any time
- ❖ The control flow is not discernible from the code
- ❖ These issues will be discussed in more detail in Section 3

Control Flow for GUI Application Systems



GUI Programs for Multiple Window Systems

- ❖ **Window systems provide developers with many features**
 - ♦ Similar in the interpretation and implementation of typical GUI objects - windows, dialogs, menus, controls, etc.
 - ♦ Different programming models - such as callbacks
 - ♦ Different data structures, functional API, and terminology
- ❖ **Different window systems present information in different graphical forms and with different screen organizations**
 - ♦ Different interface look-and-feel
 - ♦ Important consideration while designing the GUI
- ❖ **How should a developer design an application to work on different window systems?**

Introduction to XVT-Design++ Development

Slide 19

Approaches to Portability

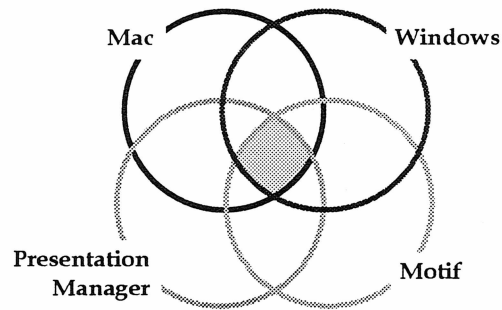
- ❖ **Least Common Denominator -**
 - ♦ Two platforms may have a lot in common
 - ♦ Seven platforms will have very little if anything in common
- ❖ **Emulation -**
 - ♦ Implement code which imitates the behavior of the native systems
 - ♦ Limited in that window systems are always changing
- ❖ **Abstraction -**
 - ♦ The XVT approach
 - ♦ A thin layered API
 - ♦ Applications are truly native

Introduction to XVT-Design++ Development

Slide 20

Approaches to Portability

❖ Least Common Denominator -



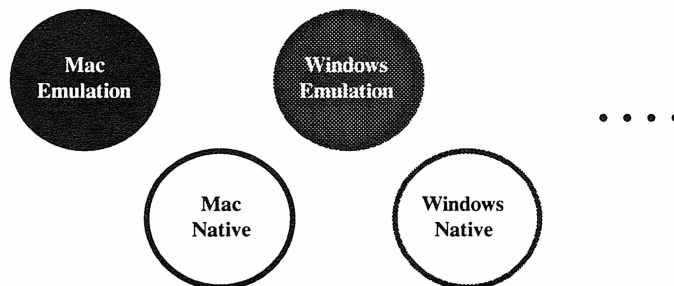
Introduction to XVT-Design++ Development

Slide 21

Approaches to Portability (cont.)

❖ Emulation - implement code which imitates the behavior of the native systems

- ♦ what if the emulated native system changes or improves?



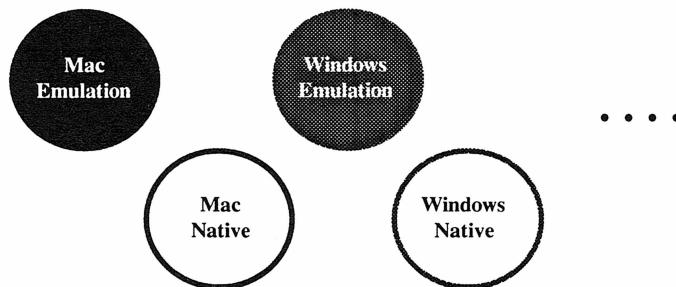
Introduction to XVT-Design++ Development

Slide 22

Approaches to Portability (cont.)

❖ Emulation - implement code which imitates the behavior of the native systems

- ♦ what if the emulated native system changes or improves?



Introduction to XVT-Design++ Development

Slide 23

Approaches to Portability - Abstraction

❖ Advantages of abstraction:

- ♦ An application written with XVT-Design++ is a native application
- ♦ You get a truly native look and feel (not emulated)
- ♦ You can easily conform to the native style guidelines and meet compliance criteria
- ♦ Any changes in the native look and feel are easier to track
- ♦ There is no danger of look and feel copyright violations
- ♦ Layered API still provides access to the native GUI toolkit
- ♦ Raising the level of abstraction provides an easier to use API

Introduction to XVT-Design++ Development

Slide 24

Portable Event Dispatching

- ❖ XVT++ classes represent the GUI object types
- ❖ XVT defines which events are applicable for each type of GUI object
- ❖ XVT++ virtual member functions defined for each event type
- ❖ You override event handling member functions to change default behavior

Introduction to XVT-Design++ Development

Slide 25

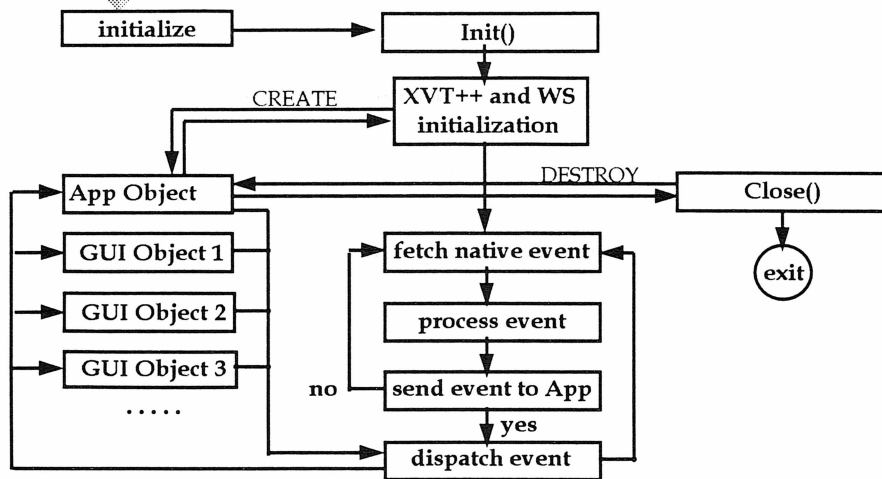
Portable Event Dispatching (cont.)

- ❖ Inside the XVT++ class library -
 - ♦ Events are received from the native event dispatcher
 - ♦ XVT++ sends the event to the GUI object within which it occurred
- ❖ At the XVT-Design++ application level -
 - ♦ The appropriate event handling member function processes the event

Introduction to XVT-Design++ Development

Slide 26

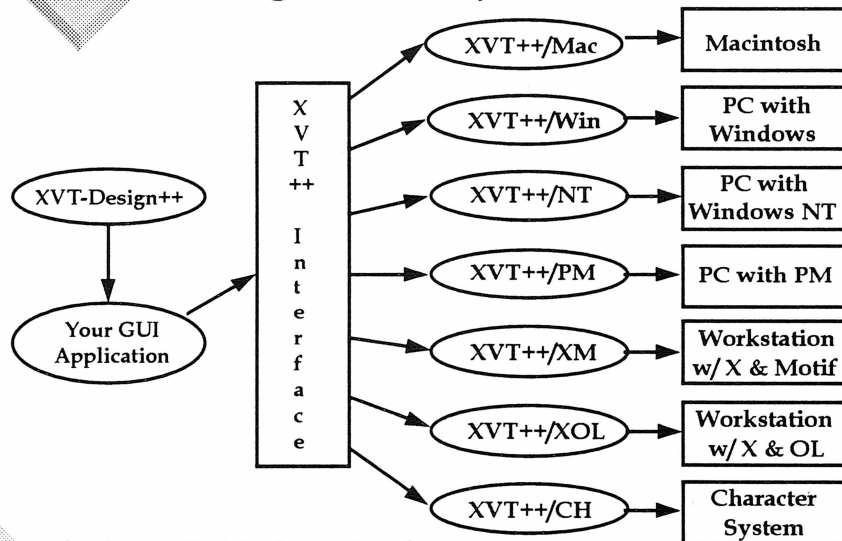
Control Flow for XVT++ Programs



Introduction to XVT-Design++ Development

Slide 27

XVT-Design++ Platforms



Introduction to XVT-Design++ Development

Slide 28

XVT-Design++ and Portability

- ❖ Using XVT-Design++ doesn't *guarantee* that every program will be portable - it *allows* it to be
- ❖ The XVT-Design++ application developer must be aware of portability issues:
 - ♦ Compiler and OS differences
 - ♦ Window system differences
 - ♦ User interface style-guideline differences
- ❖ **Prototype on each target platform early in your development, so you're not surprised**

 Introduction to XVT-Design++ Development

Slide 29

Different Native GUI Platforms - Overview

- ❖ Application object
- ❖ Menubar placement
- ❖ Window Manager influences
- ❖ Application startup and termination
- ❖ Available screen real estate and resolution

 Introduction to XVT-Design++ Development

Slide 30

The XVT Application Object

- ❖ The task window is better thought of as the application object
 - ♦ It represents the application itself
- ❖ May or may not be a physical window
 - ♦ Depending on the platform
- ❖ The task window processes events relevant to the application object
- ❖ Provides a portable solution to a *very* non-portable issue

Introduction to XVT-Design++ Development

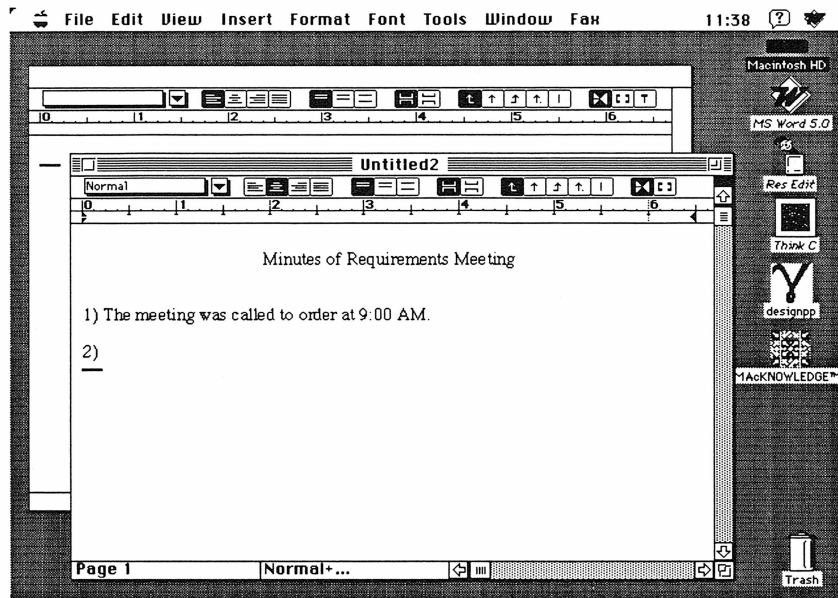
Slide 31

Macintosh and Character

- ❖ No Task Window
- ❖ Menubar-per-screen
 - ♦ Fixed location
- ❖ Minimal visual effect at startup - menubar change
- ❖ Closing all windows does not terminate the application
- ❖ Limited screen real estate
 - ♦ CH or Mac Classic
- ❖ Unique aspects
 - ♦ Macintosh "Option" Key
 - ♦ Single button mouse
 - ♦ Mouseless operation in CH is common

Introduction to XVT-Design++ Development

Slide 32



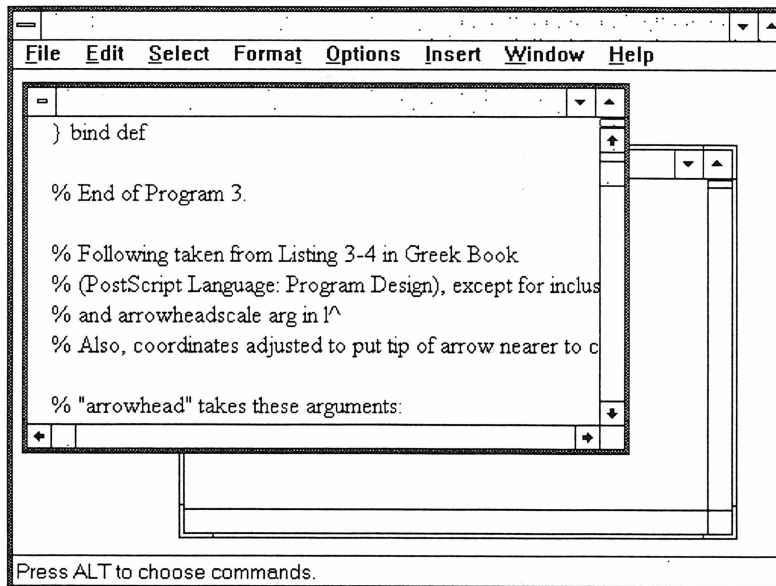
Introduction to XVT-Design++ Development

MS-Windows, NT & OS/2

- ❖ **Visible Task Window**
 - ♦ Appears on startup
 - ♦ Disappears on termination
- ❖ **Menubar-per-task window**
- ❖ **Variety of screen resolutions**
- ❖ **Unique aspects**
 - ♦ Multiple Document Interface - MDI (Windows)
 - ♦ Threaded Processes (OS/2 and NT)
 - ♦ Reserved help function key (F1)

Introduction to XVT-Design++ Development

Slide 34



Introduction to XVT Development

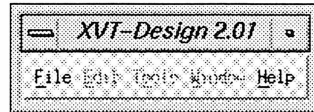
Motif & OPEN LOOK

- ❖ **No visible Task Window**
 - ♦ "Ghost window" appearance on X platforms
- ❖ **Menubar-per-window**
- ❖ **Style guidelines recommend termination when last window is closed**
- ❖ **Large, high-resolution monitors**
- ❖ **Unique aspects**
 - ♦ Variety of user-configurable window managers
 - ♦ Focus models
 - (ie: focus follows pointer vs. set focus via mouse click)
 - ♦ Pinned menus (OPEN LOOK)
 - ♦ 3 button mouse

Introduction to XVT-Design++ Development

Slide 36

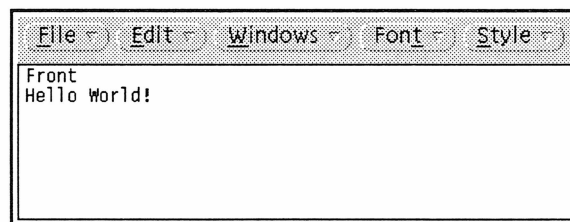
Motif - "Ghost Window"



Introduction to XVT-Design++ Development

Slide 37

Open Look



Introduction to XVT-Design++ Development

Slide 38

XVT C++ Cross-Platform Development Tools

❖ **XVT-Design++ (GUI Builder)**

- ♦ Design++/Win - Microsoft Windows
- ♦ Design++/NT - Microsoft Windows/NT
- ♦ Design++/PM - IBM OS/2 Presentation Manager
- ♦ Design++/CH - Character Displays (UNIX, DOS, VMS)
- ♦ Design++/Mac - Apple Macintosh
- ♦ Design++/XM - OSF/Motif (UNIX, etc.)
- ♦ Design++/XOL - Sun OPEN LOOK

❖ **XVT++ Class Library**

❖ **XVT Portability Toolkit (PTK)**

Introduction to XVT-Design++ Development

Slide 39

XVT-Design++

- ❖ **Preferred gateway to XVT++ programming**
- ❖ **Accelerates the construction of cross-platform XVT++ applications**
- ❖ **Speeds interface layout and development**
- ❖ **Generates GUI structure code and portable GUI resources**
- ❖ **Enables rapid prototyping and GUI testing**

Introduction to XVT-Design++ Development

Slide 40



XVT++ Class Library

- ❖ **Source-form C++ class hierarchy**
 - ♦ **Classes and Member Functions**
 - ♦ **Platform-independent**
- ❖ **Layered on top of the XVT Portability Toolkit**



Introduction to XVT-Design++ Development

Slide 41



XVT Portability Toolkits

- ❖ **Implemented as platform-specific libraries**
- ❖ **Written in C**
- ❖ **Toolkit Application Programmer's Interface (API)**
 - ♦ **Functions, Macros, Defined Constants, Data Types**
- ❖ **Universal Resource Language (URL)**
- ❖ **Compiler for URL (CURL)**
- ❖ **Other tools**
 - ♦ **ERRSCAN**
 - ♦ **CHELP**



Introduction to XVT-Design++ Development

Slide 42

Terminology - C++ vs XVT++

- ❖ **GUI Object** - a component of the interface; usually visible on the screen; includes menus, windows, dialogs, and controls
- ❖ **C++ Object** - an instance of a class; an instance of a C++ structure that encapsulates both data and code
- ❖ **GUI Container** - usually a window or dialog; these may contain other GUI objects, such as controls, graphics, or text
- ❖ **C++ Container** - an instance of a C++ class for maintaining collections of objects of arbitrary type

Introduction to XVT-Design++ Development

Slide 43

Terminology

- ❖ **Resources** - external definitions of the attributes and layout of the GUI objects
- ❖ **Menubar** - contains menus which contain menu items; usually at the top of the screen or a window
- ❖ **Window** - a rectangular portion of the screen that is used to display or gather information
- ❖ **Child Window** - positioned, moved, and contained within another (parent) window
- ❖ **Dialog** - a rectangular portion of the screen that is used to gather information using controls

Introduction to XVT-Design++ Development

Slide 44

Terminology

- ❖ Controls - specific GUI objects for gathering user input; come in a variety of styles
- ❖ Widgets - an X-Windows term for a GUI object
- ❖ Events - occurrences within the GUI that the application should respond to; may be initiated by the user, application, or system
- ❖ Tags - may be related to an event; in general, a location within a source file for action code
- ❖ Action Code - any code entered at a Tag; for example, the code that will respond to an event

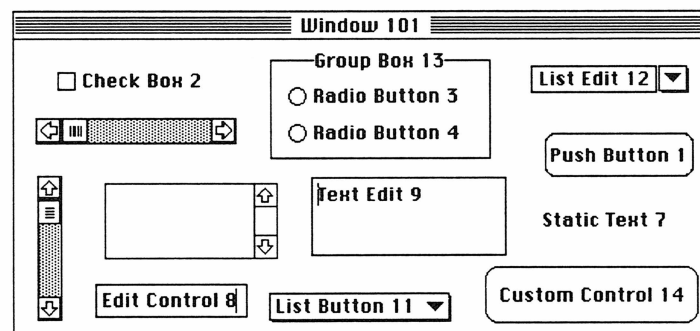
Introduction to XVT-Design++ Development

Slide 45

Terminology - Control Types

❖ Controls (XVT Terminology)

Push Button	Group Box	List Box
Check Box	Static Text	List Button
Edit Control	Radio Button	List Edit
Text Edit	Scrollbars	Custom Controls



Introduction to XVT-Design++ Development

Slide 46

Typographical Conventions

- ❖ **Code**
 - ♦ This typestyle represents code, including expressions and names of classes, member functions, attributes, and variables
- ❖ **Filenames**
 - ♦ Plain type is used for filenames and file extensions
- ❖ ***Emphasis***
 - ♦ Italics are used for emphasis and the names of documents
- ❖ **Commands**
 - ♦ This typestyle is used for command options within menus and dialogs
- ❖ **TAGS**
 - ♦ Capitals are used for the names of events and special tags

Introduction to XVT-Design++ Development

Slide 47

Course Example - Rationale

- ❖ **Course Example objective is to build and browse an employee database**
- ❖ **Process - design, implement, and compile an XVT GUI application**
- ❖ **Purpose - see how the XVT-Design++ and XVT++ functionality is used**
- ❖ **Source code will be provided**

Introduction to XVT-Design++ Development

Slide 48

Course Example - Purpose

- ❖ **Introduction to the process of building XVT++ applications**
 - ♦ using XVT-Design++ and the XVT++ Class Library
- ❖ **Insights into -**
 - ♦ The advantages of each XVT product
 - ♦ Where to best apply their strengths
- ❖ **Introduction to -**
 - ♦ XVT++ programming techniques
 - ♦ Application structuring

Introduction to XVT-Design++ Development

Slide 49

Course Example - Purpose (cont.)

- ❖ **Examine example code which implements typical GUI functionality**
- ❖ **Have a working XVT++ application as a model for future reference**
- ❖ **Some disclaimers:**
 - ♦ Not all aspects of XVT-Design++ or the XVT++ Class Libraries will be exercised
 - ♦ Application is not intended to be a reference example of interface layout or usability
 - ♦ Non-GUI functionality (the application part) supports the GUI programming in this exercise

Introduction to XVT-Design++ Development

Slide 50

Course Example - Methodology

Requirements	Section 3
Plan and Design	Section 3
Develop and Test	Sections 4 - 6
Complete and Build	Section 6

Introduction to XVT-Design++ Development

Slide 51

Course Example - Methodology

- ❖ Discuss the Course Example's need for some functionality
- ❖ Discuss the GUI concept associated with the need
- ❖ Present the XVT-Design++ features that support that need
- ❖ Use XVT-Design++ to layout the GUI objects
- ❖ Discuss how to manage the GUI object
- ❖ Use XVT-Design++ to implement the action code
- ❖ Present any API details related to the topic

Introduction to XVT-Design++ Development

Slide 52



Course Introduction

**Questions on why we are here
or what we will be doing?**



This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.



Section 2

A Tour of XVT-Design++



Introduction to XVT-Design++ Development

Slide 1



What This Section Covers

- ❖ **The Role of XVT-Design++**
- ❖ **The Role of You as Programmer**
- ❖ **XVT-Design++ Introduction**
 - ♦ Resource Layout and Edits
 - ♦ Setting Connections between GUI Objects
 - ♦ TestMode and Prototyping
 - ♦ Program Code and Makefile Generation
 - ♦ Advanced Features
- ❖ **Building a First Application - hello.prj**
- ❖ **Using XVT-Design++ and XVT++ Together**



Introduction to XVT-Design++ Development

Slide 2



The Role of XVT-Design++

- ❖ Accelerates the development of the GUI portion of XVT++ applications
- ❖ Provides default action code for selected GUI object tags
- ❖ Facility for assigning action code to a GUI object's tags - a higher level programming technique
- ❖ Particularly good for rapid prototyping of user interface design options
 - ♦ Connections and TestMode



Introduction to XVT-Design++ Development

Slide 3



The Role of XVT-Design++ (cont.)

- ❖ Generates portable application structure code and program resources, and hides the details of this code
- ❖ Automatically generates platform-specific makefiles
- ❖ Useful to have XVT-Design++ on each target platform
 - ♦ Speeds interface layout creation, testing, and changes
 - ♦ Not available for Design++/CH



Introduction to XVT-Design++ Development

Slide 4

The Role of XVT++

- ❖ **Abstraction of many native GUI toolkits**
 - ♦ portable application programmer interface (API)
 - ♦ portable external resource model (URL)
- ❖ **Layered API provides native GUI toolkit access**
 - ♦ Use XVT++ portable functionality wherever possible
 - ♦ Use non-portable functionality as needed (sparingly)
 - ♦ Non-interference with native GUI look-and-feel
- ❖ **Provides classes that represent GUI object types**
 - ♦ Provides member functions for object management and manipulation
 - ♦ Provides virtual event handler member functions for applicable events

Introduction to XVT-Design++ Development

Slide 5

Role of You as GUI Programmer

- ❖ **Define relationship between the GUI and the application code**
- ❖ **Design and layout user interface**
 - ♦ Window, dialog, and menu contents
 - ♦ Interactions between them
 - ♦ Enforce #define naming conventions
- ❖ **Build application code - database access and update, application logic**
- ❖ **Successively refine action code for events, adding links to application code**

Introduction to XVT-Design++ Development

Slide 6

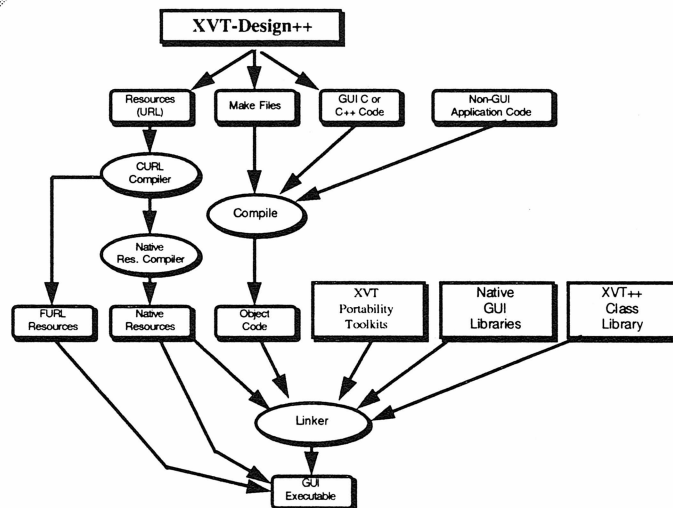
Summary of Roles

- ❖ What XVT-Design++ does for you
 - ♦ Provides framework source code
 - ♦ Provides source code for connections
 - ♦ Generates source code files, URL files, and makefiles
- ❖ What you do with XVT-Design++
 - ♦ Define and layout GUI objects
 - ♦ Set connections and use TestMode to prototype
 - ♦ Implement additional action code
- ❖ What you do outside of XVT-Design++
 - ♦ Write container classes and custom controls
 - ♦ Implement non-GUI code
 - ♦ Compile and build the application

Introduction to XVT-Design++ Development

Slide 7

Overview of XVT++ Programming



Introduction to XVT-Design++ Development

Slide 8

Key XVT-Design++ Concepts

- ❖ **Graphical, interactive interface layout**
- ❖ **XVT-Design++ is implemented using XVT products**
- ❖ **GUI object attributes**
 - ♦ Geometry, Title, Resource ID, etc.
- ❖ **Tags - place holders for action code fragments**
 - ♦ Event Tags - for each type of input the GUI may receive
 - ♦ Special Tags - ancillary code related to the GUI objects
- ❖ **Context - the combination of a Module, Object, and Tag that specifies the identify of / location for an action code fragment**

 Introduction to XVT-Design++ Development

Slide 9

Key XVT-Design++ Concepts (cont.)

- ❖ **Connections**
 - ♦ Relationships between GUI objects
 - ♦ Code generated for the connection
- ❖ **TestMode**
 - ♦ Evaluate GUI object layout and connections
- ❖ **Action Code Editor (ACE)**
 - ♦ Used to attach and write code fragments
 - ♦ Generated framework code
- ❖ **Source code generation**
 - ♦ C++ GUI code
 - ♦ External resource file (URL)
 - ♦ Makefiles

 Introduction to XVT-Design++ Development

Slide 10

XVT-Design++ Project Files

- ❖ A project file contains all information about your XVT-Design++ application
 - ♦ Project attributes
 - ♦ GUI Object Layout and Attributes
 - ♦ Action code
- ❖ Default project file names end in .prj
- ❖ File format - binary and portable
- ❖ Multiple projects can be edited at the same time
- ❖ Convert XVT-Design project files to XVT-Design++ (and vice versa)

Introduction to XVT-Design++ Development

Slide 11

Layout Editor

- ❖ Direct manipulation of GUI objects
 - ♦ "Paint" your GUI interface
- ❖ WYSIWYG layout
 - ♦ GUI objects: windows, dialogs, and controls
 - ♦ Menubars and strings have their own editors
- ❖ Layout convenience features
 - ♦ Alignment grid
 - ♦ Alignment and sizing options
- ❖ Generates portable resources with a portable layout feature

Introduction to XVT-Design++ Development

Slide 12

Other XVT-Design++ Functionality

- ❖ Container Classing feature
- ❖ Custom Controls
- ❖ Entering Help text via the application HELP tag
- ❖ Creation Order Editor
- ❖ Strings Editor
- ❖ User Data
- ❖ PFA Utility Functions

 Introduction to XVT-Design++ Development

Slide 13

XVT-Design++ GUI Objects

- ❖ **Primary GUI Objects: Containers**
 - ◆ Application Object (Task Window)
 - ◆ Windows
 - ◆ Dialogs
- ❖ **Secondary GUI Objects**
 - ◆ Controls
 - ◆ Menubars
- ❖ **NOTE: Secondary GUI Objects always contained by or attached to windows or dialogs**

 Introduction to XVT-Design++ Development

Slide 14

Windows

❖ Containers for:

- Graphic primitives (drawing)
- Controls
- Text

❖ May (optionally) have an associated menubar

❖ Usually defined via external resources with XVT-Design++

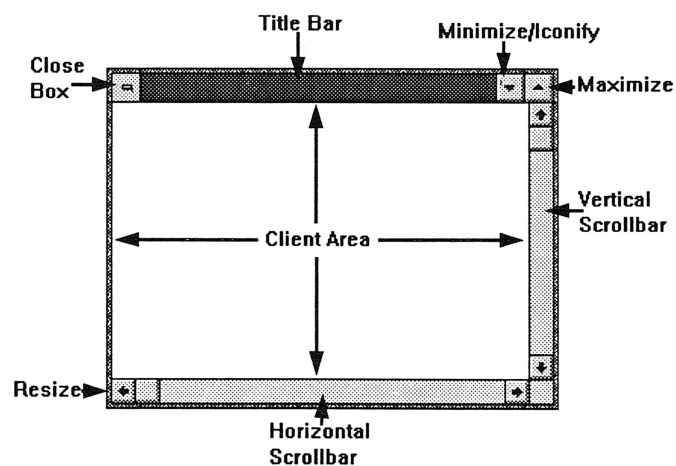
❖ Window components

Client area	Resize controls
Title bar	Border scrollbars
Close box	Maximize / minimize / iconify boxes

Introduction to XVT-Design++ Development

Slide 15

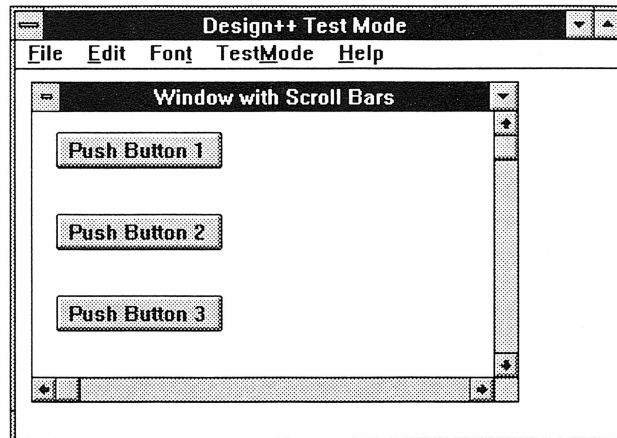
Window Components



Introduction to XVT-Design++ Development

Slide 16

Document Window with Scrollbars



Introduction to XVT-Design++ Development

Slide 17

XVT-Design++ Window Types

- ❖ Document (title bar plus some border decorations)
- ❖ Plain (single) border
- ❖ Double border

Introduction to XVT-Design++ Development

Slide 18

Dialogs

- ❖ Containers for controls only - no graphics
 - ♦ Can include icons
- ❖ Gives access to native keyboard navigation of controls
- ❖ Modality property
 - ♦ Modal (halts program execution until dismissed)
 - ♦ Modeless (program execution continues)
- ❖ Usually defined via external resources with XVT-Design++

Introduction to XVT-Design++ Development

Slide 19

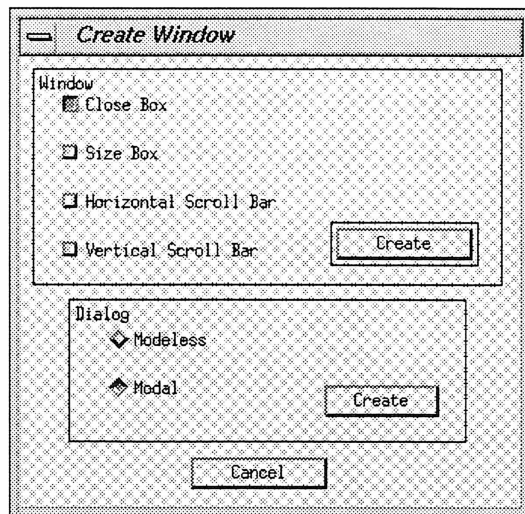
Dialogs

The screenshot shows a dialog box titled "Change Dialog Attributes". It contains several input fields and checkboxes. The "Title" field is set to "Other Choices". The "#define" field is set to "DLG_CHOICES". The "Class" field is set to "XVT_Dialog". Under the "Type" section, the "Modal" radio button is selected, while "Invisible" and "Disabled" checkboxes are unchecked. The "Modeless" radio button is also present but not selected. The "X" coordinate is 160, "Y" is 120, "Width" is 320, and "Height" is 240. "OK" and "Cancel" buttons are at the bottom right.

Change Dialog Attributes			
Title	Other Choices		Class
#define	DLG_CHOICES		XVT_Dialog
Type		<input type="checkbox"/> Invisible	
<input checked="" type="radio"/> Modal		<input type="checkbox"/> Disabled	
<input type="radio"/> Modeless			
X	160	Width	320
Y	120	Height	240
		OK	
		Cancel	

Introduction to XVT-Design++ Development

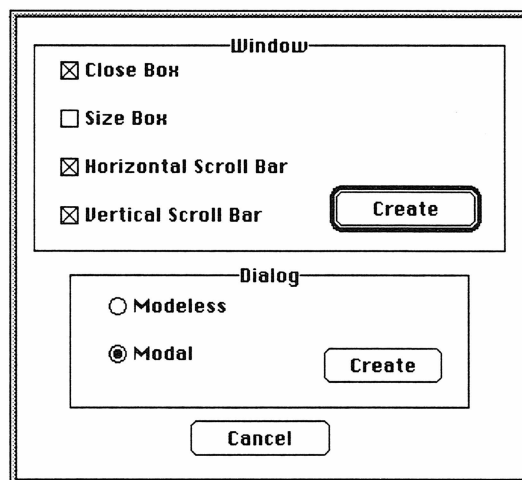
Dialogs -- Motif



Introduction to XVT-Design++ Development

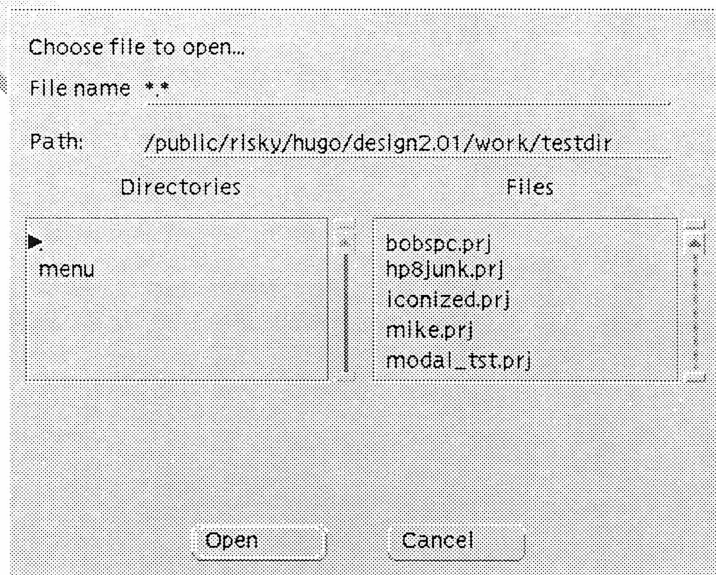
Slide 21

Dialogs - Macintosh



Introduction to XVT-Design++ Development

Slide 22



Introduction to XVT-Design++ Development

Slide 23

Dialogs (cont.)

- ❖ **Pre-defined dialogs available as a convenience**
 - ♦ Notebox and messages, opening and saving files, etc.
- ❖ **No parent-child hierarchies as with windows**
- ❖ **Can not possess menubars**

Introduction to XVT-Design++ Development

Slide 24

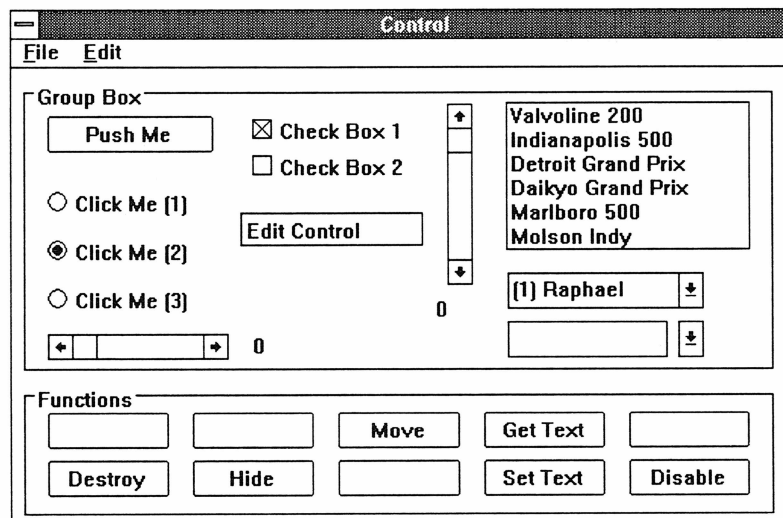
Controls

❖ Always placed within a window or dialog

❖ XVT-Design++ control types

push button	text edit
check box	list box
radio button	list button
scrollbars	list edit
static text	group box
edit control	custom control

❖ Usually defined relative to container via external resources with XVT-Design++



GUI Object Attributes

❖ Typical attributes that can be set include:

- ♦ Titles
- ♦ #define identifiers (resource ID's)
- ♦ Geometry attributes (overrides layout)
- ♦ Menu resource ID (for windows)
- ♦ GUI object "classing" (more later)

GUI Object Attributes (cont.)

- ❖ **Double-clicking on GUI objects during layout displays the attribute dialog for the GUI object**
 - ♦ You also can select the Layout or Attribute push button in the ACE for the GUI object specified by the current context
- ❖ **The attributes for the application object can be set via the Edit menu, or the Attributes... pushbutton in the ACE**
- ❖ **Attribute changes may not be immediately visible within XVT-Design++**
- ❖ **They will be visible within TestMode**

Menubars

- ❖ Property of windows (not dialogs)
- ❖ Usually defined as resources via URL with XVT-Design++
- ❖ Dynamically replaceable and modifiable
- ❖ Menu item attributes
 - ♦ Normal, checked, disabled, separator, child (for hierarchical menus), mnemonics and accelerators
- ❖ Availability of portable “standard menus” for convenience
 - ♦ File, Edit, Help, Font/Style

Menu Editor

- ❖ Tool for interactively building hierarchical menus
 - ♦ Add menu items, separators, child menus, move items
- ❖ Set menu item attributes
 - ♦ Menu item text
 - ♦ #define (for menu resource ID's)
 - ♦ Mnemonics and accelerators
 - ♦ Checkable, checked, disabled
 - ♦ Separator
- ❖ XVT-Design++ standard menus can be used

Menu Editor (cont.)

- ❖ **Unlimited number of menus can be defined - all will be reflected in the generated URL resource code**
 - ♦ Possible to create menus that your program will assign to windows during program execution
- ❖ **A task window default menubar is available**
 - ♦ Serves as a starting point for defining an application specific menubar

Connections and TestMode

- ❖ **Assign symbolic connections between certain GUI object events (tags) and GUI object actions**
 - ♦ ie: When this pushbutton is pressed, create this window.
- ❖ **Not intended to be all-encompassing, but rather a basic tool for building simple prototypes**
- ❖ **When a connection is possible, the Connections... pushbutton in the ACE is enabled**
- ❖ **Connections can only be set for the following:**
 - ♦ When the application object (task window) is created
 - ♦ When a pushbutton is pressed
 - ♦ When a menu item is selected

Connections and TestMode (cont.)

❖ **Actions that can be performed:**

- ♦ Create a user-defined GUI object (window or dialog)
- ♦ Create a predefined XVT-Design++ dialog
- ♦ Close a GUI object
- ♦ No connection

❖ **Connect to GUI object:**

- ♦ List of available containers
- ♦ List of predefined XVT-Design++ dialogs

❖ **C++ code for the connection is inserted into the GUI object's tag**

Connections and TestMode (cont.)

❖ **Connections can be exercised via TestMode**

❖ **To get started in TestMode:**

- ♦ Set a connection to the CREATE event tag for the application object (task window), or...
- ♦ Set a connection to at least one menu item in the application object (task window) menubar
- ♦ Otherwise, TestMode cannot bootstrap itself!

❖ **TestMode does not exercise source code**

Connections and TestMode - Removing Connections

- ❖ Once established, the generated code for the connection can be removed and the connection will still exist within TestMode
 - ♦ Why? Connections are implemented internally via a simple symbolic interpreter, and the generated code is not used
- ❖ Removing a connection will *not* remove the code that XVT-Design++ placed in the GUI object tag!
- ❖ Removing connections is a two step process
- ❖ To fully remove a connection:
 - ♦ Select No Connection from within the Connection... dialog
 - ♦ AND Cut the connection code from the ACE pane

Introduction to XVT-Design++ Development

Slide 35

The Action Code Editor (ACE)

- ❖ The ACE can be used to:
 - ♦ Create and edit action code
 - ♦ Create and modify connections
 - ♦ Invoke the layout and attributes editors
- ❖ The ACE can be invoked by:
 - ♦ Choosing Action Code Editor from the Tools menu
 - ♦ Choosing Edit Code from the Edit menu when the layout window for a window or dialog is active
 - ♦ Holding down the Control or Option (platform dependent) key and double-clicking on a control or layout window

Introduction to XVT-Design++ Development

Slide 36

ACE- The Editing Context

- ❖ The editing context is specified by three selections - the Module, Object, and Tag
- ❖ **Module**
 - ♦ Any of the applications containers
 - ♦ Any application menubar
 - ♦ The application object (task window)

ACE- The Editing Context (cont.)

- ❖ **Object - Any of the GUI objects contained by the item specified by the Module list button**
 - ♦ Usually controls or menu items
 - ♦ Items in the list have a prefix that indicates their type:
 - DLG: for a dialog or WIN: for a window
 - ITEM: for menu items
 - A two letter abbreviation is used for controls, e.g. PB: for pushbuttons
- ❖ **Tag - Any of the tags available for the item specified by the Object list button**
 - ♦ Event tags have the prefix EVNT:
 - ♦ Special tags have the prefix SPCL:



ACE - The Text Editing Pane

- ❖ Used to create, modify, and examine action code fragments
- ❖ The code in the pane corresponds to the current context
- ❖ The pane can be scrolled and resized (resize the ACE window)
- ❖ Code text can be edited with Cut, Copy, and Paste commands
- ❖ The font and size of the code text can also be changed
- ❖ The code is saved as part of the project file and will be included in the generated source files



Introduction to XVT-Design++ Development

Slide 39



ACE- Other Controls

- ❖ Layout - brings up the layout window for the context object
- ❖ Attributes... - opens the attributes dialog for the context object
- ❖ Default Code - inserts the default action code supplied by XVT-Design++
- ❖ Revert - discards any changes made to the code since the context was last selected



Introduction to XVT-Design++ Development

Slide 40

File Generation

❖ **XVT-Design++ will automatically generate:**

- ♦ URL files for the resources
- ♦ C++ source code files to handle the GUI objects

❖ **You specify:**

- ♦ The destination directory
- ♦ The application name
- ♦ Which files to generate
- ♦ The filenames (if the defaults are not to your liking)

File Generation - File Types

❖ **C++ source code files**

- ♦ A separate .cc file for each module
- ♦ Includes member functions and code fragments

❖ **Header files**

- ♦ A separate .h file for each module

❖ **Resource (URL) files**

❖ **Help Text files**

❖ **Makefiles**

- ♦ Template-based
- ♦ Custom templates
- ♦ The XVTDIR & XVTPPDIR macros
- ♦ External files

Building A First Application

- ❖ Let's build an "Hello World" application
- ❖ We'll follow the example from *Using XVT-Design++*
- ❖ The application will include the following features:
 - ♦ The user can open any number of windows
 - ♦ Windows display a message, specified by the user
 - ♦ Messages can be changed using menu items or radio buttons within a dialog
 - ♦ The user can select the font, size, and style for the message, using the Font menu

 Introduction to XVT-Design++ Development

Slide 43

The Application Building Process

- ❖ Start a new project and set the project attributes
- ❖ Layout the menus, windows, dialogs, and controls
- ❖ Set connections between GUI objects
- ❖ Use TestMode to evaluate the layout and connections (save the project)
- ❖ Write action code for event and special tags as dictated by the application requirements
- ❖ Generate the application (save the project)
- ❖ Build and run the application
- ❖ Return to XVT_Design++ to edit the application

 Introduction to XVT-Design++ Development

Slide 44

Review: XVT-Design++ and XVT++ Applications

❖ XVT-Design++ generates:

- ♦ Portable, XVT++ based application code
- ♦ Portable XVT++ interface resource file (URL)
- ♦ Platform-specific makefiles

❖ XVT++ Class Library: portable API programming

- ♦ Use XVT++ supplied classes to develop the GUI application's event handler member functions

Introduction to XVT-Design++ Development

Slide 45

Where do we go from here ...

❖ Plan and develop an XVT GUI application using both XVT-Design++ and the XVT++ Class Libraries

Introduction to XVT-Design++ Development

Slide 46

This image shows a single sheet of white paper with horizontal black ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Section 3

Concepts of Portable XVT-Design++ Programming

 Introduction to XVT-Design++ Development

Slide 1

What This Section Covers - The Best Way to Use XVT-Design++

- ❖ **Planning and Architecting an Application**
 - ♦ The Model-View-Controller Paradigm
 - ♦ Desktop Metaphor
 - ♦ Styles of User Interaction
- ❖ **Application Management**
- ❖ **GUI Object Creation**
- ❖ **External Interface Resources**

 Introduction to XVT-Design++ Development

Slide 2



What This Section Covers (cont.)

❖ Events

- ♦ An Overview
- ♦ GUI Object / Tag Pairs
- ♦ Processing

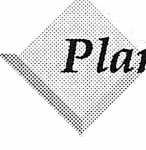
❖ Course example program

- ♦ Requirements
- ♦ Design
- ♦ Structure



Introduction to XVT-Design++ Development

Slide 3



Planning an Application

- ❖ Iterative process of plan, coding, testing during prototyping phase
- ❖ Prototyping completed to arrive at final application concept
- ❖ Desktop metaphor - application must coexist with other applications
- ❖ Different styles of user interaction



Introduction to XVT-Design++ Development

Slide 4

The GUI Desktop Metaphor

- ❖ **Multiple applications may be sharing the desktop**
- ❖ **The desktop manager handles interactions between applications**
 - ♦ Sharing of information between applications
 - ♦ Keyboard focus - Where do user keyboard events go?
- ❖ **Applications must be good citizens**
 - ♦ Share screen real estate with moveable and sizeable windows
 - ♦ Share system resources (memory allocation and release)
- ❖ **Manipulation of other applications may directly affect your application**
 - ♦ Window system shutdown requests

Introduction to XVT-Design++ Development

Slide 5

Different Styles of User Interaction

- ❖ **User manages the application via keyboard and/or mouse**
 - ♦ Keyboard-only operation?
- ❖ **Novice users**
 - ♦ Menu items and dialogs
 - ♦ Help information
- ❖ **Advanced users**
 - ♦ Accelerators and mnemonics
 - ♦ Keyboard navigation
 - ♦ Key chords and mouse interactions
- ❖ **GUI should be easy to learn and easy to use**

Introduction to XVT-Design++ Development

Slide 6

Other Planning Considerations

- ❖ Each of these must also be planned for:
 - ♦ Printing
 - ♦ Portable file handling
 - ♦ Platform-specific issues
- ❖ These and other topics will be discussed in Section 6

Introduction to XVT-Design++ Development

Slide 7

Architecting an Application

- ❖ The Model-View-Controller (M-V-C) paradigm
 - ♦ Model - Application Data and Methods
 - ♦ View - Perspective(s) on the Model provided through the GUI
 - ♦ Controller - Access control layer between View and Model
- ❖ M-V-C supports code modularization and team development
- ❖ Separates GUI Code (View and Controller) from Application Code (Model)
- ❖ GUI Applications are Event Driven
- ❖ Do not forget an Error Handling Policy

Introduction to XVT-Design++ Development

Slide 8

M-V-C Paradigm

❖ The simplest example, and one that applies to the Course Example -

- ♦ The Model consists of
 - the database data structures (classes)
 - member functions for maintaining the database
- ♦ The View and Control is provided by an XVT GUI
 - providing the user with pictures of data records
 - controlling the updating of data records based on user inputs

Introduction to XVT-Design++ Development

Slide 9

Key Technique: Separation of GUI and Application Code

❖ GUI Code (View and Controller)

- ♦ Creating/ destroying/ managing windows and dialogs
- ♦ Set controls to initial values and retrieve user input
- ♦ Respond to relevant events via event handler member functions
- ♦ Provide GUI data to application code in application's format
- ♦ Degree of direct manipulation influences of separation of View and Controller

❖ Application Code (Model)

- ♦ Application code not concerned with how it gets data, nor user interface look-and-feel and management

Introduction to XVT-Design++ Development

Slide 10

Key Technique: Separation of GUI and Application Code (cont.)

- ❖ The design for the interface between GUI and application code is best driven by application code needs
 - ♦ “The GUI serves the application” is a good approach
- ❖ Separation of GUI object definitions from GUI object management code is also desirable
 - ♦ Automatic if GUI objects defined via external resources (using URL) as with XVT-Design++
- ❖ Benefits
 - ♦ Simplifies changes to the user interface and to the application code underneath
 - ♦ Easier to split work within a programming team
 - ♦ Greatly reduces overall application code complexity

Introduction to XVT-Design++ Development

Slide 11

Structuring Event-Driven GUI Code

- ❖ The GUI code comprises the View and the Controller
- ❖ XVT-Design++ generates some of the code needed for the GUI
- ❖ You will write the rest of the GUI code, typically using the Action Code Editor (ACE)
- ❖ To do this, you must understand the *event-driven* nature of XVT-Design++ applications

Introduction to XVT-Design++ Development

Slide 12

XVT-Design++ Applications are Event-Driven Applications

- ❖ **Application structured to respond to events**
 - ♦ Member functions are called to handle events as they are received
 - ♦ Example: "Window A, you have just been resized by the user. Here is your new size."
- ❖ **Application behavior based on the events you choose to handle**
- ❖ **Application control flow depends on the order in which events occur**

Introduction to XVT-Design++ Development

Slide 13

Important: Structure Application to Relinquish Control

- ❖ **One aspect of the desktop metaphor was that multiple applications coexist**
- ❖ **An application responds to an event, then gives control back to the windowing system**
- ❖ **Applications should process events in a timely fashion, or application responsiveness degrades**
- ❖ **What program is in control? Native windowing system program!**

Introduction to XVT-Design++ Development

Slide 14

Error Handling

❖ User Errors

- ♦ Make errors easy to recover from
- ♦ XVT++ provides several informational and error dialog facilities
- ♦ Provide user with information about the error, and what they can do to correct it
- ♦ Avoid wordings that blame the user

❖ System Errors

- ♦ Develop policy of handling fatal errors
- ♦ XVT++ provides error manager and error handler classes

Introduction to XVT-Design++ Development

Slide 15

Application Management

❖ Programming teams

- ♦ Notational conventions
- ♦ XVT-Design++ projects cannot be merged programmatically
 - Cut-and-paste to merge project files
- ♦ Modularity based on M-V-C paradigm

❖ Source control

- ♦ XVT-Design++ project files are binary format
- ♦ Generated source files are more easily archived

❖ Testing - Port Early and Often

- ♦ Test on each of the hardware platforms that the final system is to be delivered

Introduction to XVT-Design++ Development

Slide 16

How to Use XVT-Design++

❖ Two approaches to building an application:

- ♦ Layout then Connections then Action Code
 - Pass over all of GUI objects on each phase
 - This is how the tutorial in the user's manual does it
- ♦ Individual GUI Objects (Menus, Windows, Dialogs)
 - Layout, connect, and code for each GUI object
 - This is how we will do it in the course

❖ The choice is a matter of style, not substance

❖ For larger applications, a table can be useful

Introduction to XVT-Design++ Development

Slide 17

How to Use XVT-Design++

GUI <u>Object</u>	Define/ <u>Layout</u>	<u>Connect</u>	Action <u>Code</u>	<u>Filename</u>
Menu1				
Menu2				
Window1				
Window 2				
Window 3				
Dialog1				
Dialog2				

Introduction to XVT-Design++ Development

Slide 18

GUI Object Creation Methods

❖ **Resource-based (via XVT-Design++ definitions)**

- ♦ Function call references GUI object by resource ID
- ♦ GUI object attributes defined externally in URL file
- ♦ Best method - use XVT-Design++

❖ **Programmatic**

- ♦ `Init()` member function call specifies creation parameters and initial GUI object attributes
- ♦ No reliance on external resource definitions
- ♦ Use of this approach has diminished because of the increased role of XVT-Design++

Introduction to XVT-Design++ Development

Slide 19

GUI Object Creation Methods - Structuring the Application

❖ **For smaller applications, there will be two modules:**

- ♦ GUI code (as generated by XVT-Design++)
- ♦ Application code (as developed using a native editor)

❖ **For larger application, there may be three modules:**

- ♦ GUI code (as generated by XVT-Design++) but with function calls to ...
- ♦ GUI code that was developed using a native editor and maintained in separate files
 - Controller code
 - Container classes
 - Custom controls
- ♦ Application code (as developed using a native editor)

Introduction to XVT-Design++ Development

Slide 20

External Interface Resources

❖ Why are external resources useful?

- ♦ GUI objects, layout, and attributes definable outside of C++ code (can be changed external to C++ code)
- ♦ Separation of GUI object definitions from code that manages the GUI objects
- ♦ GUI object definitions referred to by resource ID
- ♦ Easy to localize (change language) of user interface

❖ XVT Universal Resource Language (URL)

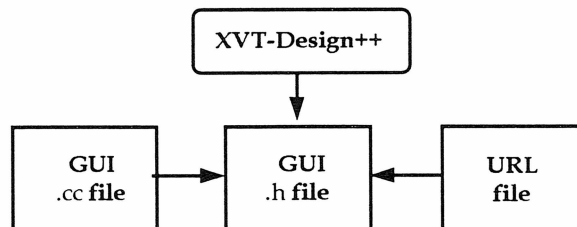
- ♦ Language (syntax) for defining windows, dialogs, controls, menubars, and strings

Introduction to XVT-Design++ Development

Slide 21

External Interface Resources (cont.)

- ❖ Communication regarding resources thru `#define` resource IDs
- ❖ Resource IDs set within XVT-Design++
- ❖ Resource ID `#define`'s located in `.h` file



Introduction to XVT-Design++ Development

Slide 22

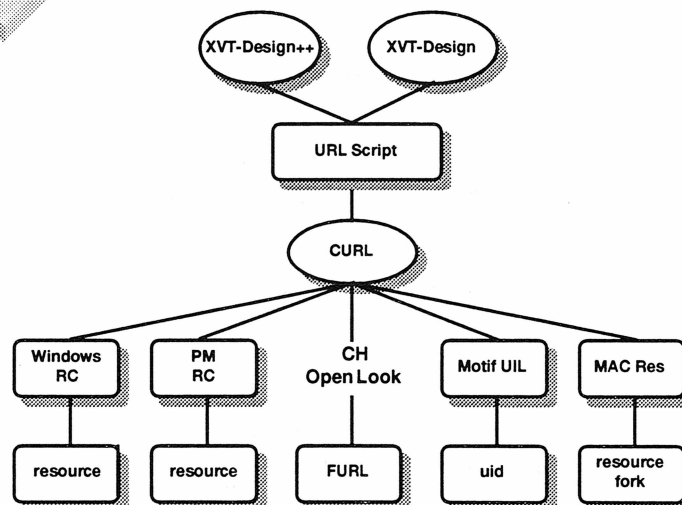
External Interface Resources (cont.)

- ❖ **Compiler for URL (CURL)**
 - ♦ Converts portable URL format to platform-specific resources
- ❖ **Creating URL files**
 - ♦ The fastest and easiest approach: XVT-Design++
 - ♦ Manual creation (time-consuming but possible to do)
- ❖ **XVT-Design++ includes a new CURL that will turn old .URLs into XVT-Design++ .PRJs for easier program maintenance**

Introduction to XVT-Design++ Development

Slide 23

URL Compilation



Introduction to XVT-Design++ Development

Slide 24

URL File Syntax

```
MENU TASK_MENUBAR_22
  ITEM MN_DB_OPEN "~Open"
  SEPARATOR
  ITEM MN_DB_BROWSE "~Browse..." DISABLED
  ITEM MN_DB_CLOSE "~Close" DISABLED

DIALOG EMPDIALOG URL_RECT(63,39,497,286) "Employee Data" MODELESS
  BUTTON ED_NEXT_PB URL_RECT(168,224,110,24) "Next" DEFAULT
  BUTTON ED_CANCEL_PB URL_RECT(312,224,110,24) "Cancel"
  EDIT ED_EMPID_EC URL_RECT(104,16,120,22) ""
  EDIT ED_FIRSTNAME_EC URL_RECT(104,48,120,22) ""
  EDIT ED_LASTNAME_EC URL_RECT(104,80,120,22) ""
  EDIT ED_TITLE_EC URL_RECT(104,112,120,22) ""
```

Portable GUI Event Representation

❖ What is an event?

- ♦ An event is a description of an input which occurs
- ♦ An event will be of a specific type
- ♦ Other information may be associated with the event

❖ When they are generated?

- ♦ Events can be generated by the user, application or system
- ♦ Events can be generated at any time in nearly any order

❖ How they are used?

- ♦ The GUI application will respond to the events
- ♦ You will write the code that determines the response

Categories of XVT Events

❖ Requests vs. Notification

- ♦ Request - user has selected a menu item
- ♦ Notification - user just resized your window

❖ Application-level events

- ♦ Sent to the application object (task window)
- ♦ Example - user has directed the application to be closed

❖ GUI Object-level events

- ♦ Events for the GUI object itself
- ♦ Sent to their own event handler member functions

Introduction to XVT-Design++ Development

Slide 27

XVT-Design++: A Higher-Level Event Model

❖ Use of GUI object-tag pairs

- ♦ Tags correspond to either GUI object events or code locations within a module
- ♦ They represent attachment points for GUI action code

❖ Each type of GUI object receives a different set of events

❖ Each type of GUI object also has associated with it a subset of the special tags

Introduction to XVT-Design++ Development

Slide 28

Events: GUI Object-Tag Pairs

- ❖ Automatic construction and generation of event handler member function code
- ❖ Virtual member function for each event type
- ❖ Use the Action Code Editor (ACE) to fill out code for event and special tags

Introduction to XVT-Design++ Development

Slide 29

GUI Object-Event Tag Pairs

Task	Window		Menu	Dialog		Controls
	<u>Window</u>	<u>Window</u>	<u>Item</u>	<u>Dialog</u>	<u>Controls</u>	
Action			X			X
Char		X		X		
Close	X	X		X		
Create	X	X		X		X
Destroy	X	X		X		X
Focus		X		X		X (some)
Font	X	X				
*scroll		X				
Mouse *		X				
Quit	X					
Size	X	X		X		
Timer	X	X		X		
Update		X				
User	X	X		X		X

Introduction to XVT-Design++ Development

Slide 30

GUI Object-Special Tag Pairs

	Task		Menu		
	<u>Window</u>	<u>Window</u>	<u>Item</u>	<u>Dialog</u>	<u>Controls</u>
Help	X				
Is Quit OK	X				
Main Code	X				
User Header	X				
User URL	X				
Class Decl	X	X	X	X	X
Class Header		X		X	
Top	X	X		X	
Bottom	X	X		X	
Constructor	X	X	X	X	X
Destructor	X	X	X	X	X

Introduction to XVT-Design++ Development

Slide 31

GUI Development Guidelines

- ❖ Principle of least astonishment
- ❖ Consistency within and between applications
- ❖ User conventions, work flow, and visual metaphors
- ❖ Standard menu layouts, mnemonics, operations, etc.
- ❖ Feedback, warnings, confirmation, error checking, etc.
- ❖ Rules for control layout, grouping, and usage
- ❖ Color displayed and perceived uniquely

Introduction to XVT-Design++ Development

Slide 32

GUI Development Guidelines

- ❖ Often platform dependent - you will probably need to structure your code accordingly
- ❖ Emphasis the ease of GUI learning / use
- ❖ Consider work flow analysis to learn more about how the GUI will be used
- ❖ In all cases, conduct prototype user testing to validate and refine the GUI

Introduction to XVT-Design++ Development

Slide 33

Key Technique: Port Early and Often

- ❖ Determine and acquire target platforms early in development cycle
- ❖ Test task window (application object) implementations on various XVT-Design++ platforms
- ❖ Layout and geometry differences (especially with controls)
- ❖ C++ code portability "gotchas" discovered early
- ❖ Build up expertise in the use of each native platform

Introduction to XVT-Design++ Development

Slide 34

Course Example - Functional Requirements

❖ What outputs does the system need to provide?

- ♦ The ability to open and browse a database
- ♦ The ability to review and update database records
- ♦ A graphical display of an employee's salary history
- ♦ A interactive gatherer of timesheet information

❖ Where do the inputs come from?

- ♦ An existing database whose name and location will be specified by the user
- ♦ The user will also provide record update information

Course Example - Functional Requirements (cont.)

❖ Process Control

- ♦ The user has complete control over the process
- ♦ They will specify the database, control the display of records, and call up ancillary information as desired
- ♦ Once a database is open, the user will want to switch between adjacent records

Course Example - Design

❖ Translating requirements into GUI objects and events

- ♦ Outputs are assigned to windows or dialogs
- ♦ Inputs come from menus, dialogs, and controls
- ♦ Process control is via menus and dialogs

❖ Define each GUI object by its:

- ♦ Contents, layout, attributes, etc.
- ♦ Provide screen shots for user review
- ♦ Better yet, demonstrate a prototype

Course Example - Design (cont.)

❖ The course example program will need:

- ♦ A window to display the database summary
- ♦ A dialog to present an employee's record
- ♦ A window to display the salary history
- ♦ A dialog to present and gather timesheet information

❖ For each of these, decide what command options need to be made available - controls and/or menu items

❖ Layout (with XVT-Design++) the physical representation of each of these containers



Section Summary

- ❖ **Planning an application**
 - ♦ Architecture and management
- ❖ **GUI object creation and external resources**
 - ♦ #define resource IDs
- ❖ **GUI object / tag pairs**
 - ♦ Event and special tags

- ❖ **On to building an application ...**

[illegible]



Section 4

Building an XVT-Design++ Application

 Introduction to XVT-Design++ Development

Slide 1

What This Section Covers

- ❖ **Creating an Application**
- ❖ **Application Objects/Task Windows**
- ❖ **Menubars, Menus, Submenus, and Menu Items**
- ❖ **Dialogs and Controls**
- ❖ **Connections and TestMode**
- ❖ **The Action Code Editor**

 Introduction to XVT-Design++ Development

Slide 2

Course Structure

❖ During Sections 4 and 5:

- ♦ Discuss the need for some functionality within the Course Example
- ♦ Present the GUI concept background
- ♦ Use XVT-Design++ to layout the GUI objects
- ♦ Use XVT-Design++ to connect and test the GUI objects
- ♦ Present GUI object manipulation discussion
- ♦ Use ACE to implement the action code
- ♦ Present any additional details related to the concept
- ♦ Summarize the concept

Introduction to XVT-Design++ Development

Slide 3

Course Structure (cont.)

❖ Additional details may include:

- ♦ A summary of the class' member functions
 - Syntax and Semantics
 - Inherited "utility" functions
- ♦ Remaining event and special tags
 - Discussion of purpose and use
- ♦ XVT++ role
 - What XVT-Design++ doesn't do

Introduction to XVT-Design++ Development

Slide 4

Project Creation

- ❖ **Creating the project file**
- ❖ **Setting the project attributes**
 - ♦ About box
 - ♦ Task menubar
 - ♦ Task window title
 - ♦ Document prefix
- ❖ **Setting the project file name and location**
- ❖ **Saving the project file**

Introduction to XVT-Design++ Development

Slide 5

Setting Project Attributes

The screenshot shows a dialog box titled "Project Attributes". It contains four main sections: "Task Menu Bar" with a text field containing "TASK_MENUBAR"; "Task Window Title" with a text field containing "Task Window"; "Document Prefix" with a text field containing "Document Title"; and "About Box" with a text field containing "Default". At the bottom right, there are "OK" and "Cancel" buttons.

Introduction to XVT-Design++ Development

Slide 6

The `main()` function in XVT-Design++ Applications

- ❖ Required function as in other C++ applications
- ❖ Perform application-level initializations using `MAIN CODE` special tag
- ❖ Application object `Init()` method performs system initialization

Introduction to XVT-Design++ Development

Slide 7

Performing Application Initialization

- ❖ Initializations may be GUI or non-GUI (open files, network connections, etc.)
- ❖ Non-GUI initializations
 - ♦ In `main()`, before control is relinquished to XVT++
 - ♦ Better yet, in the application object's `CREATE` event handler member function
- ❖ Perform GUI initializations in the application object's `CREATE` event handler member function
- ❖ Different methods are required for reporting errors in `main()` vs. during the application object's `CREATE` event

Introduction to XVT-Design++ Development

Slide 8

XVT-Design++ Generates main ()

```
int main( int argc, char *argv[] )
{
    XVT_Config xdConfig( TASK_MENUBAR, 0, "dbdemo", "Document
    Title", "Task Window" );
    XVTAppTaskwin *task_win = new XVTAppTaskwin;

    // Non-GUI initialization code inserted
    // here using Main Code tag

    task_win->Init( argc, argv, 0L, xdConfig );
    return 0;
}
```

Performing Application Termination

- ❖ **Call `XVT_TaskWin::Close()` to terminate your application**
- ❖ **Call in response to a program-specific event that signals that the user wants to close the application**
 - ♦ Example: User selects Quit from the File menu
 - ♦ An application can use any event as a signal to terminate
- ❖ **Call in response to event from Desktop Manager that it wants to close your application**
 - ♦ Example: User closes the Program Manager in MS Windows

Application Level Concepts

- ❖ Screen Window
- ❖ Task Window (aka Application Object)
- ❖ Task Window Events

Introduction to XVT-Design++ Development

Slide 11

Application Concepts: The Screen Window

- ❖ XVT++ class XVT_ScreenWin
- ❖ Represents the physical display screen
- ❖ `GetOuterRect()` provides the physical screen extent
- ❖ No physical appearance (other than the screen itself)
- ❖ Maintains a list of the applications top-level windows and dialogs

Introduction to XVT-Design++ Development

Slide 12

Application Concepts: The Task Window

- ❖ XVT++ class XVT_TaskWin
- ❖ Represents the “Application Object”
- ❖ May be visibly represented on some platforms
- ❖ Will always have a menubar
- ❖ Will be visible if no window with menubar is available

Introduction to XVT-Design++ Development

Slide 13

Application Concepts: Application Object Events

- ❖ Events (usually) relevant to the application object:
 - ♦ CREATE - the application has been created
 - ♦ DESTROY - the application is about to be terminated
 - ♦ QUIT - from the window manager
 - ♦ CLOSE - request to close physical task window
- ❖ Note: Other events may be received if a non-portable application object has been specified (such as a drawable task window under MS Windows)
- ❖ For the rest of this section, assume a portable (standard) application object model

Introduction to XVT-Design++ Development

Slide 14

Events: Handling CREATE Events

- ❖ First event received by the application object
- ❖ A notification event - the GUI application is running
- ❖ No arguments to the CREATE member function
- ❖ You usually perform some or all of these operations:
 - ♦ Application initialization - GUI error messages available
 - ♦ Create an initial window or display an About box
 - ♦ Store state information in the C++ object

Introduction to XVT-Design++ Development

Slide 15

Events: Handling DESTROY Events

- ❖ Final event received by the application object
- ❖ Notification that the application is terminating
- ❖ No arguments to the DESTROY member function
- ❖ Limit your work here:
 - ♦ Release resources and application finalization by, for example, disconnecting from networks, freeing allocated memory, etc

Introduction to XVT-Design++ Development

Slide 16

Events: Handling CLOSE Events

- ❖ Request event: the user has operated the close control on the border of the task window
- ❖ No arguments to the CLOSE member function
- ❖ XVT-Design++ generates default code to call `Is_Quit_OK()` and then `Close()` based on the return value
- ❖ Add code to the IS QUIT OK special tag to:
 - ♦ Save and close files or ask user to discard work, if necessary
 - ♦ Return TRUE or FALSE, as appropriate
- ❖ Calling `Close()` generates a DESTROY event (where termination code will be executed)

Introduction to XVT-Design++ Development

Slide 17

Events: Handling QUIT Events

- ❖ Received only by the application object
- ❖ Window Manager wants to terminate your application
- ❖ May either be a request or a notification - window system dependent
- ❖ If a request (`xdQuery` is TRUE), call `QuitOK()` to indicate this application approves of shutdown
- ❖ If a notification (`xdQuery` is FALSE), call `Close()`

Introduction to XVT-Design++ Development

Slide 18

XVT-Design++'s QUIT Code

```
void XVTAppTaskwin::e_quit( BOOLEAN   xdQuery )
{
    if (xdQuery)
    {
        if (is_quit_OK())
            QuitOK();
    }
    else
        Close();
}
```

Introduction to XVT-Design++ Development

Slide 19

Course Example -Application Object Action Code

- ❖ **USER HEADER special tag code**
 - ♦ Error handler class defined
- ❖ **CREATE event tag code**
 - ♦ Instantiate error handler class
 - ♦ Create initial windows
- ❖ **Default action code for**
 - ♦ CLOSE - is it OK to close
 - ♦ QUIT - handles system manager quit requests
 - ♦ MAIN CODE - create the task window

Introduction to XVT-Design++ Development

Slide 20

Application Object Details - Additional Event Tags

- ❖ **USER** - an application-defined event has been initiated
- ❖ **TIMER** - the timer associated with the Task Window has gone off
- ❖ **SIZE** - the size of the Task Window has been set or changed
- ❖ **FONT** - the user chose an item from the Font menu, the default is to call `SetFontMenu ()`

Introduction to XVT-Design++ Development

Slide 21

Application Object Details - Additional Special Tags

- ❖ **CONSTRUCTOR** - code placed within it
- ❖ **DESTRUCTOR** - code placed within it
- ❖ **TOP** - place to add declarations, constants, etc. to .cc
- ❖ **BOTTOM** - place to add member functions to .cc
- ❖ **CLASS DECL** - place to declare data and functions
- ❖ **USER_HEADER** - code placed in application .h
- ❖ **USER_URL** - code placed in URL file
- ❖ **HELP** - help text associated with the application
- ❖ **MAIN CODE** and **IS QUIT OK** - discussed previously

Introduction to XVT-Design++ Development

Slide 22

Application Object Details - Member Function Summary

- ❖ `virtual void Init(int argc, char* argv[],
unsigned long flags, XVT_Config config)`
- ❖ `void Close()`
- ❖ `virtual void e_close()`
- ❖ `virtual void e_create()`
- ❖ `virtual void e_destroy()`
- ❖ `virtual void e_quit(BOOLEAN query)`
- ❖ `virtual void e_size(XVT_Rct boundary)`
- ❖ `virtual void e_timer(XVT_Timer* timer)`
- ❖ `virtual long e_user(long id, void* data)`
- ❖ `virtual void e_font(XVT_Font font, FONT_PART
part)`
- ❖ `void QuitOK()`

Introduction to XVT-Design++ Development

Slide 23

Application Object Details - Member Function Summary (cont.)

- ❖ `virtual XVT_Rct GetInnerRect()`
- ❖ `virtual XVT_Rct GetOuterRect()`
- ❖ `EVENT_MASK GetEventMask() const`
- ❖ `XVT_ChildBase* GetFirstWin()`
- ❖ `XVT_ChildBase* GetNextWin()`
- ❖ `long GetWinCount() const`
- ❖ `void SetEventMask(EVENT_MASK mask)`
- ❖ `XVT_Menu* GetMenu()`
- ❖ `void SetMenu(XVT_Menu* menu)`
- ❖ `BOOLEAN GetTitle(char* buffer,
unsigned long* len) const`
- ❖ `void SetTitle(const char* str)`
- ❖ `void SetFontMenu(XVT_Font font)`

Introduction to XVT-Design++ Development

Slide 24

Application Object Summary

- ❖ Screen windows
- ❖ Task windows which are represented by application objects
- ❖ Key tags - CREATE, QUIT, CLOSE, and DESTROY events

Introduction to XVT-Design++ Development

Slide 25

An Application's Need for Menus

- ❖ Offer the user a set of program control options (usually grouped by subject)
- ❖ Minimal screen real estate used
- ❖ The set of control options available may differ based on the current View
- ❖ Menu items can be selected:
 - ♦ Manually with the mouse or keyboard arrow keys (novice)
 - ♦ Via mnemonics or accelerators (expert, or keyboard-only)

Introduction to XVT-Design++ Development

Slide 26

Mnemonics and Accelerators

- ❖ Accelerators are keychord equivalents of a menu items
- ❖ Presented to the user next to the menu item

- ❖ Mnemonics specify a letter key that may be typed instead of using the mouse to select a menu or menu item
- ❖ Indicated by underlining or bold-facing the appropriate letter of the menu or menu item

Introduction to XVT-Design++ Development

Slide 27

XVT-Design++ Menu Components

- ❖ Menubars
 - ♦ Represents the entire menu definition for a window
- ❖ Menus and Submenus
 - ♦ Groups of related command options
 - ♦ May have mnemonics
- ❖ Menu Items
 - ♦ Represent each individual command option in a menu
 - ♦ May simply be separators
 - ♦ May be disabled or checkable
 - ♦ May cause other submenus to appear
 - ♦ Selectable and result in ACTION events
 - ♦ May have mnemonics and/or accelerators

Introduction to XVT-Design++ Development

Slide 28

Course Example - Menu Requirements

- ❖ Utilize standard menus where possible
- ❖ Default action code for standard menu items
 - ♦ Quit, About, Help
- ❖ Application-specific menu items
 - ♦ Access, Browse, Disconnect
 - ♦ Grouped within a Database menu

Introduction to XVT-Design++ Development

Slide 29

XVT-Design++: Defining Menus

- ❖ Use XVT-Design++ Menubar Editor to create new and modify standard menus
- ❖ Menubars are window attributes
 - ♦ The same menubar may be attached to more than one window
 - ♦ Separate menubars may be defined for each window
- ❖ Task Window menubar ID passed to `Init()` in `main()`

Introduction to XVT-Design++ Development

Slide 30



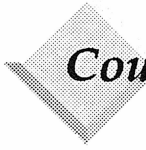
Standard XVT-Design++ Menus

- ❖ **File**
- ❖ **Edit**
- ❖ **Font/Style**
 - ♦ Generates Font events
- ❖ **Help**
 - ♦ Automatically calls Help() member function



Introduction to XVT-Design++ Development

Slide 31



Course Example -Define Menus

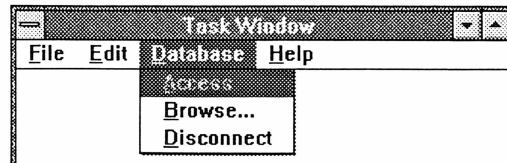
- ❖ **Menubars - one, modified TASK_MENUBAR**
- ❖ **Menus -**
 - ♦ Standard menus minus Edit and Font menus
 - ♦ New Database menu
- ❖ **Menu Items - Access, Browse, and Disconnect within Database menu**
- ❖ **Initial enabled/disabled states**
 - ♦ Set with XVT-Design++
 - ♦ Disabled all but Quit menu item within File menu
 - ♦ Enabled menu items within Database menu



Introduction to XVT-Design++ Development

Slide 32

Course Example: Change Task Menubar in Application



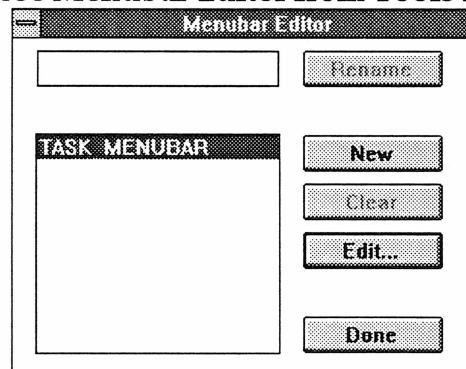
- ❖ Affects all windows that use TASK_MENUBAR
- ❖ Will contain 4 items
 - ♦ Standard File menu
 - ♦ Standard Edit menu
 - ♦ Database menu (Access, Browse..., Disconnect)
 - ♦ Standard Help menu

Introduction to XVT-Design++ Development

Slide 33

Course Example: Using the Menubar Editor

- ❖ Choose Menubar Editor from Tools menu



- ❖ Select Edit to modify the TASK_MENUBAR and bring up the Menu Editor

Introduction to XVT-Design++ Development

Slide 34

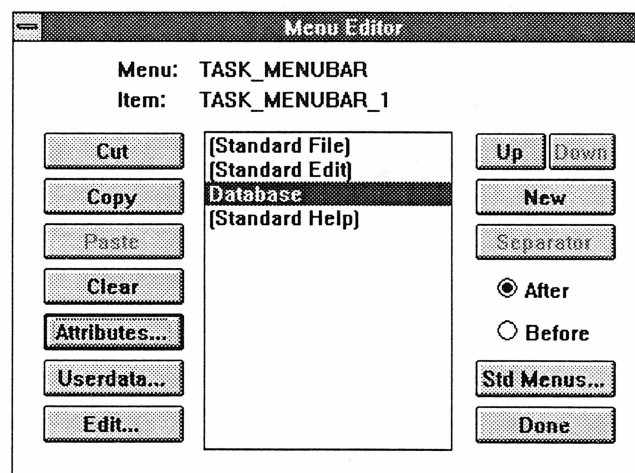
Course Example: Modify Application's Version of TASK_MENUBAR

- ❖ Clear Standard Font menu by selecting (Standard Font) in list box and press *Clear* pushbutton
- ❖ Add new Database menu by pressing *New* pushbutton
- ❖ Change name from New to Database by double clicking in list box or press *Attributes* pushbutton to change title attribute

Introduction to XVT-Design++ Development

Slide 35

Course Example: Menu Editor



Introduction to XVT-Design++ Development

Slide 36

Class Example: Adding Items to Database Menu

- ❖ With Database selected press *Edit* pushbutton
- ❖ Select *New* 3 times to add 3 new buttons to the menu
- ❖ Select each and change it's attributes
 - ♦ Title #define Mnem
 - ♦ Access MN_DB_ACCESS A
 - ♦ Browse... MN_DB_BROWSE B
 - ♦ Disconnect MN_DB_DISCONNECT D

Introduction to XVT-Design++ Development

Slide 37

Class Example: Editing Menu Items

Menu Attributes

Title: Browse...

#define: MN_DB_BROWSE

Mnemonic: B

Accelerator:

☐ Checkable

☐ Checked

☒ Disabled

☐ Alt

☐ Control

☐ Shift

Keys...

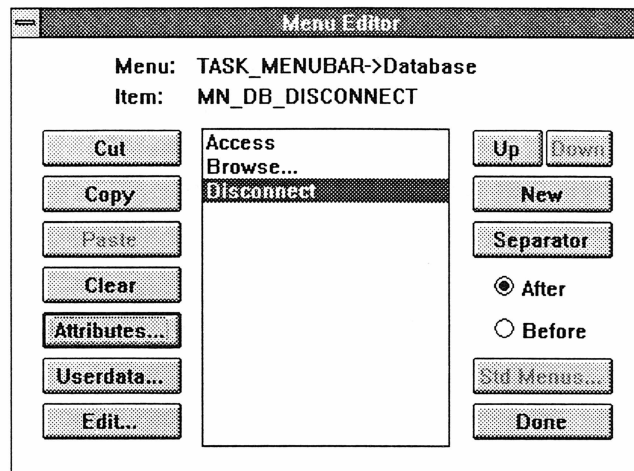
OK

Cancel

Introduction to XVT-Design++ Development

Slide 38

Class Example: Adding Items to Database Menu (Cont.)



Introduction to XVT-Design++ Development

Slide 39

Course Example -Connect Menus

- ❖ Eventually we will connect the new menu items to other GUI objects
- ❖ For now, we use a pre-defined dialog to stub out the functionality
 - ♦ Pre-defined dialogs will be discussed later in the Section
 - ♦ Stubbed out Access, Browse, and Disconnect menu items

Introduction to XVT-Design++ Development

Slide 40

Menus: Handling ACTION Events

- ❖ Request- Menu selection has been made from a window's menu
- ❖ Menu item selection identified via callback event to a menu item instance
- ❖ Arguments to event handler member function indicate if the shift or control keys were pressed
- ❖ Simple command from user - just execute appropriate application action code

Introduction to XVT-Design++ Development

Slide 41

Course Example - Menu Action Code

- ❖ Eventually, the Database menu items will:
 - ♦ Open/close other GUI objects
 - ♦ Invoke database functionality
- ❖ Those GUI objects and that code does not currently exist
- ❖ Previously we set connections for the Database menu item ACTION tags

Introduction to XVT-Design++ Development

Slide 42

Manipulating Menu Items

- ❖ Every window class has a protected data member
`XVT_Menu *Menu`
- ❖ Retrieve an `XVT_MenuNode` with
`XVT_Menu::GetItem()`
- ❖ Enable and disable menu items with
`XVT_MenuNode::SetEnabledState()`
- ❖ Check and uncheck a checkable menu item via
`XVT_MenuItem::SetCheckedState()`
- ❖ You may want to store pointers to `XVT_MenuItem` instances in your derived window classes
- ❖ Get a menu item's title with
`XVT_MenuNode::GetTitle()`

Introduction to XVT-Design++ Development

Slide 43

Manipulating Menus

- ❖ Existing menus can be retrieved from a window via `XVT_MenuWin::GetMenu()`
- ❖ A menu can be set into a window via
`XVT_MenuWin::SetMenu()`
- ❖ Add an `XVT_MenuNode` to an `XVT_Menu` using
`XVT_Menu::Install()`

Introduction to XVT-Design++ Development

Slide 44

Menus Details: Additional Tags

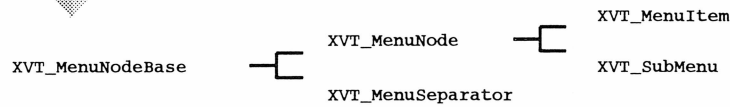
❖ **Special tags:**

- ❖ **CLASS DECL** - place to declare instance data
- ❖ **CONSTRUCTOR** - code that will be placed in the C++ object's constructor
- ❖ **DESTRUCTOR** - code that will be placed in the C++ object's destructor

XVT_Menu: Member Function Summary

- ❖ `void Install(XVT_MenuNodeBase* node)`
- ❖ `void Replace(XVT_MenuItem* item)`
- ❖ `long GetCount() const`
- ❖ `XVT_MenuNodeBase* GetFirst()`
- ❖ `XVT_MenuNodeBase* GetNext()`
- ❖ `XVT_MenuNode* GetItem(MENU_TAG tag)`

Menu Items: Member Function Summary



- ❖ XVT_MenuNodeBase
 - ♦ XVT_Menu* GetParent()
- ❖ XVT_MenuNode
 - ♦ BOOLEAN GetEnabledState() const
 - ♦ void SetEnabledState(BOOLEAN state)
 - ♦ short GetMKey() const
 - ♦ BOOLEAN GetTitle(char* buffer, unsigned long len) const
 - ♦ void SetTitle(char* str)

Introduction to XVT-Design++ Development

Slide 47

Menu Items: Member Function Summary (cont.)

- ❖ XVT_SubMenu
 - ♦ XVT_Menu* GetSubMenuPtr() const
 - ♦ void SetSubMenuPtr(XVT_Menu* submenu)
- ❖ XVT_MenuItem
 - ♦ virtual void e_action(BOOLEAN shift, BOOLEAN control)
 - ♦ BOOLEAN GetCheckableState() const
 - ♦ BOOLEAN GetCheckedState() const
 - ♦ void SetCheckedState(BOOLEAN state)

Introduction to XVT-Design++ Development

Slide 48

Menus: Summary

- ❖ Menubars attached to windows
- ❖ Menubars comprised of menu, submenus, and menu items
- ❖ XVT provides standard menus
- ❖ Key tags - ACTION events

Course Example - Dialog Requirements

- ❖ System requirement
 - ♦ Browse database records
- ❖ What information will be gathered?
 - ♦ Which record to review - next or previous
 - ♦ Employee information - ID, name, and title
 - ♦ Which information to review - salary history or timesheet
 - ♦ Type of employee - salary, hourly, or temporary
 - ♦ Type of insurance
 - ♦ Type of benefits - automatic deposit or health club
 - ♦ Whether to OK or cancel updates

Dialogs: Concept

- ❖ Dialogs are Controllers (in the M-V-C paradigm)
- ❖ Dialogs collect input from the user needed to complete an application command or update the Model
- ❖ OK, Cancel, and/or Escape controls are typical

Dialogs: Characteristics

- ❖ Special purpose: GUI container object for controls
- ❖ Modality attribute (modal or modeless)
- ❖ Support native platform keyboard navigation between controls
- ❖ Border decorations depend on the native system style
- ❖ Cannot be used to display graphical output (text or images); cannot have menubars

XVT-Design++: Defining Dialogs

- ❖ Use New Dialog **menu item** in **Windows menu**
- ❖ **Dialog attributes include:**
 - ♦ Title
 - ♦ Initial size and location
 - ♦ Modality
 - ♦ Visibility and enabledness
 - ♦ Class
 - ♦ #define resource ID
- ❖ **Layout individual controls - more in a moment**

Introduction to XVT-Design++ Development

Slide 53

Pre-Defined Dialogs

- ❖ XVT_ASK - Ask the user a yes-or-no question
- ❖ XVT_NOTE - Display an informational message
- ❖ XVT_ERROR or XVT_MESSAGE - Display an Alert or Error message
- ❖ OPEN_FILE_DLG or SAVE_FILE_DLG - Prompt the user for a file name for input or output

- ❖ Available thru the XVT-Design++ Connections... dialog

Introduction to XVT-Design++ Development

Slide 54

Other Pre-Defined Dialogs

- ❖ To put up the application About box, use
`XVT_GlobalAPI::About ()`
- ❖ To get a string typed by the user, call
`XVT_GlobalAPI::Response ()`

Course Example - Dialog Layout and Connections

- ❖ **Dialog layout and attributes**
 - ♦ Dialog is Modeless
 - ♦ Connection in Browse menu item ACTION tag to open the dialog
- ❖ **OK pushbutton**
 - ♦ (Temporary) connection in ACTION tag to close the dialog



Dialogs: Handling CREATE Events

- ❖ Notification that the dialog has been created
- ❖ The first event received by a dialog
- ❖ No additional arguments are passed
- ❖ XVT-Design++ will instantiate the controls
- ❖ Opportunity to initialize dialog controls
 - ♦ Check boxes, radio buttons, list buttons, edit controls, etc.



Dialogs: Handling CLOSE Events

- ❖ Request by the user to close the dialog
- ❖ No additional arguments are passed
- ❖ XVT-Design++ default action code is to call `Close()`
- ❖ Should exhibit the same response as when the user selects a Cancel button

Dialogs: Handling DESTROY Events

- ❖ Notification that the dialog has been closed
- ❖ The last event received by a dialog
- ❖ No additional arguments are passed
- ❖ Free any memory or resources used by the dialog

Dialogs: Special Tags

- ❖ CLASS DECL - place to declare data and functions
 - ♦ Store a pointer to the data object or a working copy of it
- ❖ CLASS HEADER - place to add #include's to .h file
- ❖ TOP - place to add declarations, constants, etc.
- ❖ BOTTOM - place to define member functions
- ❖ CONSTRUCTOR - place to add code to constructor
 - ♦ Typically initializes the data members
- ❖ DESTRUCTOR - place to add code to destructor

Dialogs: Details

❖ **Keyboard navigation**

- ♦ Use Creation Order dialog from the Edit menu to specify visit order

❖ **Mnemonics**

- ♦ Platform-dependent implementation
- ♦ Place tildes in control title before character
- ♦ XVT-Design++ does not represent

❖ **Modal dialogs**

- ♦ Do not forget OK and Cancel push buttons

❖ **Remember -**

- ♦ OK or Cancel push buttons have no automatic behavior
- ♦ Dialogs must be closed via `Close()`

Introduction to XVT-Design++ Development

Slide 61

Dialogs - Additional Event Tags

- ❖ **FOCUS** - the dialog has gained or lost keyboard focus
- ❖ **CHAR** - the user pressed a key on the keyboard and a control that receives CHAR events did not have focus
- ❖ **SIZE** - the dialog size was set or changed
- ❖ **TIMER** - a timer associated with the dialog went off
- ❖ **USER** - an application-defined event was initiated

Introduction to XVT-Design++ Development

Slide 62

Dialogs - Member Function Summary

- ❖ `BOOLEAN Init(long rid)`
- ❖ `BOOLEAN Init(WIN_TYPE wtype, XVT_Rct boundary, const char *title, long flags)`
- ❖ `void Close()`
- ❖ `virtual void e_char(short chr, BOOLEAN shift, BOOLEAN control)`
- ❖ `virtual void e_close()`
- ❖ `virtual void e_create()`
- ❖ `virtual void e_destroy()`
- ❖ `virtual void e_focus(BOOLEAN active)`
- ❖ `virtual void e_size(XVT_Rct boundary)`
- ❖ `virtual void e_timer(XVT_Timer* timer)`
- ❖ `virtual void e_user(long id, void* data)`
- ❖ `BOOLEAN GetEnabledState() const`
- ❖ `EVENT_MASK GetEventMask() const`

Introduction to XVT-Design++ Development

Slide 63

Dialogs - Member Function Summary (cont.)

- ❖ `XVT_Control* GetCtl(long cid)`
- ❖ `long GetCtlCount() const`
- ❖ `XVT_Control* GetFirstCtl()`
- ❖ `XVT_Control* GetNextCtl()`
- ❖ `BOOLEAN GetTitle(char* buffer, unsigned long* len) const`
- ❖ `BOOLEAN GetVisibleState() const`
- ❖ `void SetEnabledState(BOOLEAN state)`
- ❖ `void SetEventMask(EVENT_MASK mask)`
- ❖ `void SetInnerRect(XVT_Rct boundary)`
- ❖ `void SetTitle(const char* str)`
- ❖ `void SetVisibleState(BOOLEAN state)`
- ❖ `virtual XVT_Rct GetInnerRect() const`
- ❖ `virtual XVT_Rct GetOuterRect() const`

Introduction to XVT-Design++ Development

Slide 64

Dialogs: Summary

- ❖ Dialogs are Controllers and/or Views
- ❖ Dialogs contain controls
- ❖ Dialogs are Modal or Modeless
- ❖ Initializing dialogs and their controls
- ❖ XVT provides pre-defined dialogs
- ❖ Dialogs support keyboard navigation
- ❖ Can not draw within dialogs

Introduction to XVT-Design++ Development

Slide 65

Course Example - Controls Requirements

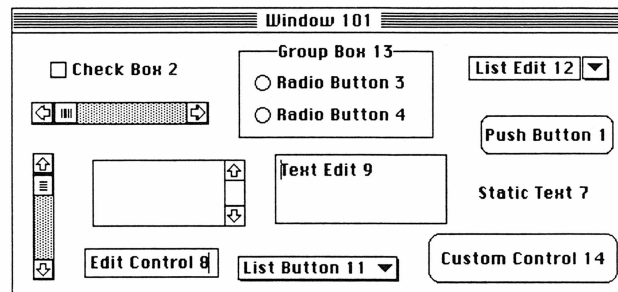
- ❖ What information will be gathered thru dialogs?
 - ♦ Which record to review - next or previous
 - ♦ Employee information - ID, name, and title
 - ♦ Which information to review - salary history or timesheet
 - ♦ Type of employee - salary, hourly, or temporary
 - ♦ Type of insurance
 - ♦ Type of benefits - automatic deposit or health club
 - ♦ Whether to OK or cancel updates
- ❖ How to map those to control types?
- ❖ How to layout the controls within the dialog?

Introduction to XVT-Design++ Development

Slide 66

Control Types

Push Button	Group Box	List Box
Check Box	Static Text	List Button
Edit Control	Radio Button	List Edit
Text Edit	Scrollbars	Custom Controls



Introduction to XVT-Design++ Development

Slide 67

Controls: Characteristics

- ❖ Each control designed to support a specialized form of user-application interaction
- ❖ Each control type is represented by a base class
- ❖ Each control added to a container is represented by a class derived from the base class
- ❖ Some degree of look-and-feel consistency across platforms

Introduction to XVT-Design++ Development

Slide 68

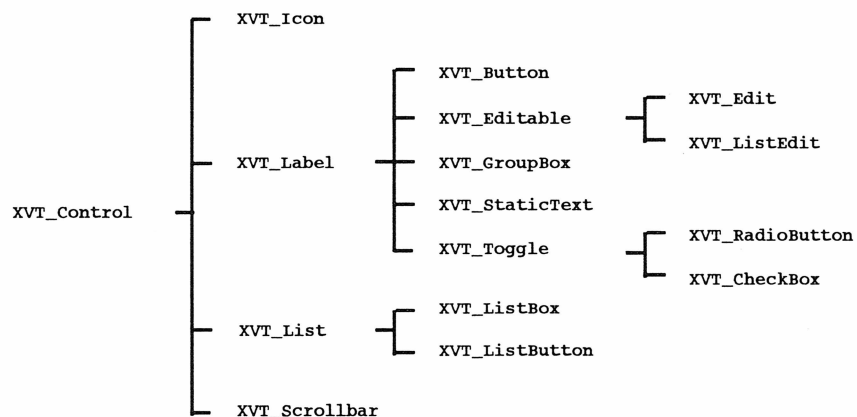
Controls: Characteristics (cont.)

- ❖ Controls always placed within a window or dialog container
- ❖ Some controls used for annotation only (no interaction)
 - ♦ Static text
 - ♦ Group boxes
 - ♦ Icons (sometimes)

Introduction to XVT-Design++ Development

Slide 69

Controls: Class Hierarchy



Introduction to XVT-Design++ Development

Slide 70

Controls: Creating Controls

❖ **Defined in external resources**

- ♦ C++ object instantiated within the `e_create()` member function of the container
- ♦ GUI object created automatically when the container is created
- ♦ C++ object destroyed automatically

❖ **Programmatically**

- ♦ C++ object instantiated at your discretion
- ♦ GUI object created with a call to `Init()`
- ♦ C++ object destroyed automatically

Controls: Managing Controls

❖ **Retrieve the `XVT_Control` object from the `XVT_Dialog` or `XVT_DrawableContainer` member function `GetCtl()`**

❖ **Controls can also be enabled/disabled and visible/invisible**

❖ **Other member functions allow you to get or set the title, get the container object, and get or set the size**

XVT_Control: Member Function Summary

- ❖ virtual BOOLEAN Init()
- ❖ virtual void Close()
- ❖ virtual void e_create()
- ❖ virtual void e_destroy()
- ❖ virtual void e_user(long id, void* data)
- ❖ BOOLEAN GetEnabledState() const
- ❖ long GetID() const
- ❖ XVT_Base* GetParent() const
- ❖ BOOLEAN GetVisibleState() const
- ❖ void MakeFront()
- ❖ void SetEnabledState(BOOLEAN state)
- ❖ void SetInnerRect(XVT_Rct boundary)
- ❖ void SetVisibleState(BOOLEAN state)
- ❖ virtual XVT_Rct GetInnerRect() const
- ❖ virtual XVT_Rct GetOuterRect() const

Introduction to XVT-Design++ Development

Slide 73

Course Example - Control Layout and Connections

- ❖ **Employee Data dialog controls**
 - ♦ Layout and Attributes
 - ♦ Grid and Align functionality
- ❖ **Stubbed out connections to undefined objects**
 - ♦ Between dialog and salary history window
 - ♦ Between dialog and timesheet window

Introduction to XVT-Design++ Development

Slide 74

Course Example: Creating the Employee Data Dialog

The screenshot shows a dialog box titled "Employee Data". It contains several input fields and controls:

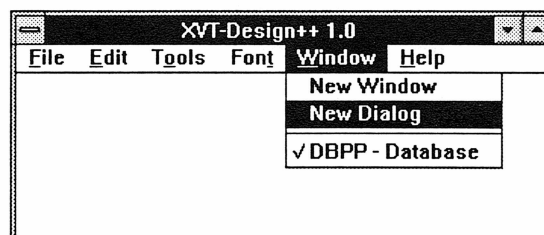
- Employee Id: 466-34-8873
- First Name: Melinda
- Last Name: Wilson
- Title: President
- Type: A group box containing three radio buttons: "Salary" (selected), "Hourly", and "Temp."
- Insurance: Plan A (with a dropdown arrow)
- Auto-Deposit: ☒ (checked)
- Health Club: ☐ (unchecked)
- Buttons at the bottom: "Salary History", "Timesheet", "Previous", "Next", "OK", and "Cancel".

Introduction to XVT-Design++ Development

Slide 75

Course Example: Creating the Employee Data Dialog

- ❖ Select *New Dialog* from the Window menu
- ❖ Setup a grid for the dialog
- ❖ Size to taste



Introduction to XVT-Design++ Development

Slide 76

Course Example: Creating the Employee Data Dialog

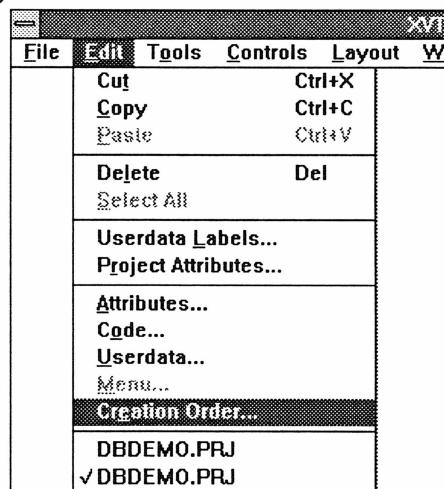
- ❖ Fairly densely populated dialog
 - ♦ 5 static text controls
 - ♦ 4 edit controls
 - ♦ 6 pushbutton controls
 - ♦ 3 radio button controls
 - ♦ 2 checkbox controls
 - ♦ 1 groupbox control
 - ♦ 1 listbutton control
- ❖ Select controls from the control menu, and place in dialog
- ❖ Set control attributes of titles and #defines

Introduction to XVT-Design++ Development

Slide 77

Course Example: Employee Data Dialog - Setting Creation Order

- ❖ From Edit menu
- ❖ Select Creation Order

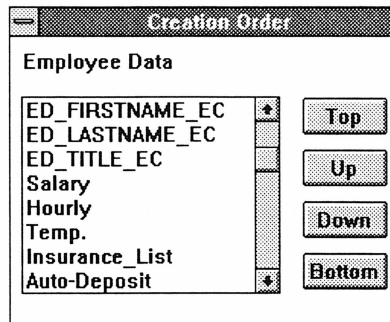


Introduction to XVT-Design++ Development

Slide 78

Course Example: Setting Creation or Tab Navigation Visit Order

- ❖ Select an item in the list, then using the *Top*, *Up*, *Down*, and *Bottom* buttons, position the item in the creation order



Introduction to XVT-Design++ Development

Slide 79

Controls: ACTION Events

- ❖ Notification that user operated a control in a container (window or dialog)
- ❖ Arguments provided to the event handler member function differ by control type -
 - ♦ For example, list boxes are told if an item was double-clicked

Introduction to XVT-Design++ Development

Slide 80



Controls: ACTION Events (cont.)

- ❖ **Controls that receive no event information**

- ♦ Push button, check box, radio button, list button, edit control, list edit

- ❖ **Controls that do receive event information**

- ♦ List box, scrollbars

- ❖ **Controls that generate no events**

- ♦ Static text, group boxes, icons



Controls: Discussion by Type

- ❖ **For each control type, we will present:**

- ♦ Introduction
- ♦ Initialization protocol
- ♦ Handling events
- ♦ Member function summary



Controls: Push Buttons

- ❖ XVT++ class XVT_Button
- ❖ **Attributes**
 - ♦ Mnemonics
 - ♦ Default
 - ♦ Visible and/or Enabled
- ❖ **Layout**
 - ♦ Consistent height and width
 - ♦ Align horizontally and vertically
- ❖ **Used for actions that move the user beyond that dialog box**
 - ♦ Use ellipses after the title when going to another dialog



Introduction to XVT-Design++ Development

Slide 83



Controls: Push Button Events

- ❖ Application-specific response to an ACTION event
- ❖ OK and Cancel push buttons say that the user is done with the dialog
 - ♦ Command complete
 - ♦ Time to update Model



Introduction to XVT-Design++ Development

Slide 84

XVT_Button Member Function Summary

- ❖ virtual BOOLEAN Init(XVT_Rct boundary, long flags = 0L, char* title = NULL)
- ❖ virtual void e_action()
- ❖ void GetTitle(char* str, unsigned long* len)
- ❖ void SetTitle(char* str)

Introduction to XVT-Design++ Development

Slide 85

Controls: Edit Fields

- ❖ XVT++ class XVT_Edit
- ❖ Attributes
 - ♦ Visible and/or enabled
- ❖ Allow the user to input a text string
- ❖ Vary in appearance and behavior by platform
 - ♦ But always one line high

Introduction to XVT-Design++ Development

Slide 86

Controls: Edit Field Events

- ❖ **ACTION event implies that an edit field's contents changed**
- ❖ **Password style of user input - store (do not display) text after each ACTION event**
- ❖ **XVT++ *will* manage the contents of an edit field**

Controls: Edit Field Events (cont.)

- ❖ **FOCUS event implies that the focus was gained or lost**
- ❖ **Focus loss - validate the contents of the edit field**
- ❖ **Focus gain - select all of the text for the user as a convenience**

- ❖ **You can query or change the contents via
XVT_Label::GetTitle() and
XVT_Label::SetTitle(), and set the current
selection via XVT_Editable::SelectText()**

XVT_Edit Member Function Summary

- ❖ virtual BOOLEAN Init(XVT_Rct boundary, long flags = 0L, char* title = NULL)
- ❖ virtual void e_action()
- ❖ virtual void e_focus(BOOLEAN active)
- ❖ void SelectText(long first, long last)
- ❖ BOOLEAN GetTitle(char* buffer, unsigned long* len)
const
- ❖ void SetTitle(const char* str)

Introduction to XVT-Design++ Development

Slide 89

Controls: List Edits

- ❖ XVT++ class XVT_ListEdit
- ❖ Attributes
 - ♦ Visible and/or enabled
- ❖ Basically an edit field with a list of candidate inputs
- ❖ Use when at least some inputs are known, but the user may enter a original input

Introduction to XVT-Design++ Development

Slide 90

Controls: List Edit Events

- ❖ List edit controls are treated like edit field controls which have an additional feature
 - ♦ A list of strings that can be selected and placed in the edit field
- ❖ Set the contents of a list component with the `XVT_List Edit` member functions
- ❖ The edit field portion of the list edit is automatically updated when the user makes a selection from the drop-down list

Introduction to XVT-Design++ Development

Slide 91

Controls: List Edit Events (cont.)

- ❖ ACTION event implies that an list edit's contents changed
- ❖ FOCUS event implies that the focus was gained or lost
- ❖ Potentially add original entries to the list
 - ♦ Keep only the most recent entries in the list

Introduction to XVT-Design++ Development

Slide 92

Controls: List Edit Events (cont.)

- ❖ Manipulate the edit field component in *exactly* the same manner as for edit field controls
- ❖ The following things can be done to a list edit control using `XVT_ListEdit` member functions:
 - ♦ Select text in the edit field component
 - ♦ Edit text in the edit field component
 - ♦ Pick an item from the list component (thus, changing the contents of the edit field component)

XVT_ListEdit Member Function Summary

- ❖ `void Add(long index, const char* str)`
- ❖ `void Add(const char* str)`
- ❖ `void Add(XVT_StrList* list)`
- ❖ `virtual BOOLEAN Init(XVT_Rct boundary, long flags = 0L, char* title = NULL)`
- ❖ `virtual void e_action()`
- ❖ `virtual void e_focus(BOOLEAN active)`
- ❖ `void SelectText(long first, long last)`
- ❖ `BOOLEAN GetTitle(char* buffer, unsigned long* len) const`
- ❖ `void SetTitle(const char* str)`

Controls: Check Boxes

- ❖ **XVT++ class XVT_CheckBox**
- ❖ **Attributes**
 - ♦ Checked, visible, and/or enabled
- ❖ **Use check boxes where a combination of selections is allowed**

Introduction to XVT-Design++ Development

Slide 95

Controls: Handling Check Box Events

- ❖ **ACTION event is a request to check or uncheck the control**
- ❖ **Default action code calls XVT_CheckBox member function SetCheckedState() to check or uncheck the control**
 - ♦ GetCheckedState() used to obtain the current state

Introduction to XVT-Design++ Development

Slide 96

XVT_CheckBox Member Function Summary

- ❖ `virtual BOOLEAN Init(XVT_Rct boundary, long flags = 0L, char* title = NULL)`
- ❖ `virtual void e_action()`
- ❖ `void SetCheckedState(BOOLEAN state)`
- ❖ `virtual BOOLEAN GetCheckedState()`
- ❖ `void GetTitle(char* buffer, unsigned long* len) const`
- ❖ `void SetTitle(const char* str)`

Controls: Radio Buttons

- ❖ **XVT++ class XVT_RadioButton**
- ❖ **Attributes**
 - ♦ Checked, visible, and/or enabled
- ❖ **Similar to check boxes, in that they can be checked or unchecked**
- ❖ **Different in that of a group, only one button will be checked at a time**
- ❖ **Group defined using the XVT++ class XVT_RadioBtnGroup**

Controls: Radio Button Groups

- ❖ Radio buttons belong to a group
- ❖ XVT-Design++ assumes that consecutively defined radio buttons comprise a group
- ❖ To be clearer, add a Group Box control before the first radio button
- ❖ XVT-Design++ generates code for the group
 - ♦ Instantiates `XVT_RadioBtnGroup` class
 - ♦ Default action code checks one button and unchecks the others

Introduction to XVT-Design++ Development

Slide 99

Controls: Radio Button Events

- ❖ ACTION event is a request to change the checked member of a group
- ❖ `XVT_Toggle::SetCheckedState()` will check or uncheck a radio button
- ❖ `XVT_Toggle::GetCheckedState()` will tell you if a radio button is checked or unchecked

Introduction to XVT-Design++ Development

Slide 100

XVT_RadioButton Member Function Summary

- ❖ virtual `BOOLEAN Init(XVT_Rct boundary, long flags = 0L, char* title = NULL)`
- ❖ virtual `void e_action()`
- ❖ `void SetCheckedState()`
- ❖ virtual `BOOLEAN GetCheckedState()`
- ❖ `void GetTitle(char* buffer, unsigned long* len) const`
- ❖ `void SetTitle(const char* str)`

Controls: Group Boxes

- ❖ **XVT++ class XVT_GroupBox**
- ❖ **Attributes**
 - ◆ Visible and/or enabled
- ❖ **Used to group controls, usually radio buttons**
- ❖ **Appears on the screen as a rectangle around the controls**
- ❖ **For annotation purposes only; no interaction capability or subsequent events**

XVT_GroupBox Member Function Summary

- ❖ virtual BOOLEAN Init(XVT_Rct boundary, long flags = 0L,
const char* title = NULL)
- ❖ BOOLEAN GetTitle(char* buffer, unsigned long* len) const
- ❖ void SetTitle(const char* str)

Controls: List Boxes

- ❖ XVT++ class XVT_ListBox
- ❖ Attributes
 - ♦ Set to support single or multiple selections
 - ♦ Can be set to be read-only
 - ♦ Visible and/or enabled
- ❖ Allow users to make single or multiple selections
from a scrollable list of candidate selections

Controls: List Box Initialization

- ❖ **Initialize a list box by:**
 - ♦ Invoking the `XVT_List::Add()` member function
 - ♦ Using `XVT_List::SetSelectedState()` to check an item
- ❖ **Prevent needless screen updates by bracketing all `XVT_List::Add()` calls within a pair of calls to `XVT_ListBox::SetSuspendedState()`**
- ❖ **Practical limit: 150 list items for portability**

Controls: List Box Events

- ❖ **ACTION event implies that an element in a list box was selected**
 - ♦ The `dbl_click` argument is `TRUE` if a double-click was performed; otherwise, it is `FALSE`
- ❖ **A single click implies a selection**
- ❖ **A double click implies a selection and some button press**

XVT_ListBox Member Function Summary

- ❖ virtual BOOLEAN Init(XVT_Rct boundary, long flags = 0L, const char* title = NULL)
- ❖ BOOLEAN Add(long index, const char* str)
- ❖ BOOLEAN Add(const char* str)
- ❖ BOOLEAN Add(long index, XVT_StrList* list)
- ❖ BOOLEAN Add(XVT_StrList* list)
- ❖ BOOLEAN Delete(long index)
- ❖ BOOLEAN Clear()
- ❖ virtual void e_action(BOOLEAN dbl_click)

XVT_ListBox Member Function Summary (cont.)

- ❖ long CountAll() const
- ❖ long CountSelections()
- ❖ XVT_StrList GetAll() const
- ❖ BOOLEAN GetElement(long index, char* buffer, unsigned long* len)
- ❖ BOOLEAN GetFirstSelection(char* buffer, unsigned long* len) const
- ❖ BOOLEAN GetSuspendedState() const
- ❖ BOOLEAN GetSelectedState(long index) const
- ❖ long GetSelectionIndex() const
- ❖ XVT_StrList GetSelections() const
- ❖ void SetSelectedState(long index, BOOLEAN select)
- ❖ void SetSuspendedState(BOOLEAN state)



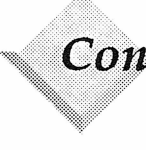
Controls: List Buttons

- ❖ XVT++ class `XVT_ListButton`
- ❖ **Attributes**
 - ♦ Visible and/or enabled
- ❖ Like a list box control but it uses less screen
- ❖ Manipulate just like a list box using the `XVT_List` member functions



Introduction to XVT-Design++ Development

Slide 109



Controls: List Button Events

- ❖ **ACTION event implies that a selection was made from a list button**
 - ♦ The button title is automatically updated
 - ♦ Dropping down the list generates no event
- ❖ **No additional arguments are provided; use `XVT_List::GetSelectionIndex()` to determine which item in the list was selected**



Introduction to XVT-Design++ Development

Slide 110

XVT_ListButton Member Function Summary

- ❖ virtual BOOLEAN Init(XVT_Rct boundary, long flags = 0L,
const char* title = NULL)
- ❖ BOOLEAN Add(long index, const char* str)
- ❖ BOOLEAN Add(const char* str)
- ❖ BOOLEAN Add(long index, XVT_StrList* list)
- ❖ BOOLEAN Add(XVT_StrList* list)
- ❖ BOOLEAN Delete(long index)
- ❖ BOOLEAN Clear()
- ❖ virtual void e_action()

Introduction to XVT-Design++ Development

Slide 111

XVT_ListButton Member Function Summary (cont.)

- ❖ long CountAll() const
- ❖ long CountSelections()
- ❖ XVT_StrList GetAll() const
- ❖ BOOLEAN GetElement(long index, char* buffer,
unsigned long* len)
- ❖ BOOLEAN GetFirstSelection(char* buffer, unsigned
long* len) const
- ❖ BOOLEAN GetSelectedState(long index) const
- ❖ long GetSelectionIndex() const
- ❖ XVT_StrList GetSelections() const
- ❖ void SetSelectedState(long index, BOOLEAN select)

Introduction to XVT-Design++ Development

Slide 112

Initializing List Boxes and List Buttons from Resources

- ❖ Retrieve strings and string lists from your resource file via the `XVT_GlobalAPI` member function `ListResStrings()`
 - ♦ Technique to avoid having hardcoded strings in your C++ code
 - ♦ Use XVT-Design++ to define resource strings

Introduction to XVT-Design++ Development

Slide 113

Controls: Static Text

- ❖ XVT++ class `XVT_StaticText`
- ❖ Attributes
 - ♦ Visible and/or enabled
- ❖ Used to label other controls
- ❖ Receive no events

Introduction to XVT-Design++ Development

Slide 114

XVT_StaticText Member Function Summary

- ❖ `BOOLEAN Init()`
- ❖ `BOOLEAN Init(XVT_Rct boundary, long flags = 0L,
const char* title = NULL)`
- ❖ `BOOLEAN GetTitle(char* buffer, unsigned long* len)
const`
- ❖ `void SetTitle(const char* str)`

Introduction to XVT-Design++ Development

Slide 115

Course Example - Action Code for Controls

- ❖ Controls of the same type have similar (if not identical) action code fragments
- ❖ Radio buttons - make dirty and check
- ❖ Check boxes - make dirty and check
- ❖ Edit - make dirty
- ❖ List button - make dirty
- ❖ Push buttons:
 - OK - update and close
 - Cancel - close
 - Next and Previous - switch records
 - Salary History - invoke function
 - Time Sheet - connection

Introduction to XVT-Design++ Development

Slide 116



Controls: Scroll Bars

- ❖ XVT++ class XVT_ScrollBar
- ❖ Attributes
 - ♦ Visible and/or enabled
- ❖ Combine with text boxes to set numeric values
- ❖ Add to text edit objects (Section 6) to allow scrolling thru large bodies of text
- ❖ Also used as sliders, volume control, and equalizers
- ❖ Initializing - set range first, then set the position and proportion



Introduction to XVT-Design++ Development

Slide 117



Controls: Scroll Control Events

- ❖ ACTION event implies that the user operated a scrollbar control
- ❖ No automatic behavior by XVT-Design++
 - ♦ No scrolling of any kind is performed
 - ♦ You must manually manage the scrollbar itself once the event is received
- ❖ Typical response:
 - ♦ Scroll or adjust whatever data is manipulated by the control
 - ♦ Then adjust the scrollbar thumb position



Introduction to XVT-Design++ Development

Slide 118

XVT_ScrollBar Member Function Summary

- ❖ virtual BOOLEAN Init(XVT_Rct boundary, long flags = 0L)
- ❖ virtual void e_action(SCROLL_CONTROL what, short pos)
- ❖ long GetScrollPosition() const
- ❖ long GetScrollProportion() const
- ❖ void GetScrollRange(long* min, long* max) const
- ❖ void SetScrollPosition(long pos)
- ❖ void SetScrollProportion(long prop)
- ❖ void SetScrollRange(long min, long max, long pos)

Introduction to XVT-Design++ Development

Slide 119

Controls: Icons

- ❖ XVT++ class XVT_Icon
- ❖ Not defineable with XVT-Design++
- ❖ Allows you to display platform-specific icons in dialogs and windows
- ❖ Defined non-portable / referred to portably
- ❖ Icons receive no events
 - ♦ Therefore no need to subclass, instantiate directly
- ❖ Member function summary
 - ♦ virtual BOOLEAN Init()
 - ♦ virtual BOOLEAN Init(XVT_Rct boundary, long cid = 0L, long flags = 0L)

Introduction to XVT-Design++ Development

Slide 120

Controls: Using XVT_StrList

- ❖ The `XVT_StrList` class specifies the interface to general-purpose lists of strings
- ❖ Each string is associated with a long word (32 bits) that can hold any data you want
- ❖ Used most commonly to move lists of strings to or from list-oriented controls
 - ♦ List boxes
 - ♦ List buttons
 - ♦ List edits
- ❖ Also returned by the `XVT_GlobalAPI` member functions `ListFaces()` and `ListResStrings()`

 Introduction to XVT-Design++ Development

Slide 121

Controls: Using XVT_StrList (cont.)

- ❖ Add a string (along with a long word of data) with the `XVT_StrList` member functions `SlistAdd()` or `SlistAddSorted()`
- ❖ Remove a string with `XVT_StrList::Remove()`
- ❖ You can retrieve the string and data by numeric index (starting with 0) with `XVT_StrList::GetElement()`
- ❖ You can loop through all of the strings with the `XVT_StrList` member functions `GetFirst()` and `GetNext()`
- ❖ You can count the number of elements in an SLIST with `XVT_StrList::Count()`

 Introduction to XVT-Design++ Development

Slide 122

XVT_StrList Member Function Summary

- ❖ void Add(long element, const char* str, long data = 0L)
- ❖ void Add(const char* str, long data = 0L)
- ❖ void Add(XVT_StrList* sl)
- ❖ void AddSorted(const char* str, long data = 0L, BOOLEAN unique = FALSE, BOOLEAN case_sensitive = FALSE)
- ❖ long Count()
- ❖ void Debug()
- ❖ void GetElement(long index, const char** str, long* data)
const
- ❖ void GetFirst(const char** str, long* data)
- ❖ void GetNext(const char** str, long* data)
- ❖ void Remove(long index)

Introduction to XVT-Design++ Development

Slide 123

Course Example: Using the Strings Editor for Insurance List

Employee Data

Employee Id: 466-34-8873

First Name: Melinda

Last Name: Wilson

Title: President

Type:
☒ Salary
☐ Hourly
☐ Temp.

Insurance:
Plan A
P.P.O.
Plan A
Plan B

☒ Auto-Deposit ☐ Health Club

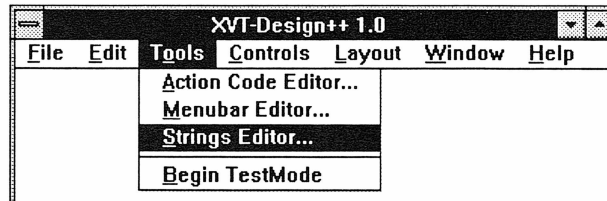
Salary History Previous OK
Timesheet Next Cancel

Introduction to XVT-Design++ Development

Slide 124

Course Example: Using the Strings Editor

- ❖ Select *Strings Editor...* from Tools menu

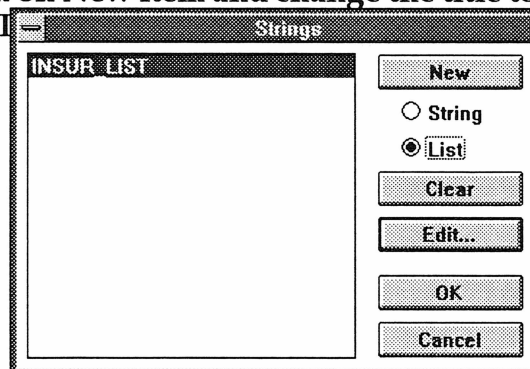


Introduction to XVT-Design++ Development

Slide 125

Course Example: Using the Strings Editor (Cont.)

- ❖ Select *List* and then select *New*
- ❖ Double click on *New* item and change the title to `INSUR_LIST`
- ❖ Select *Edit*

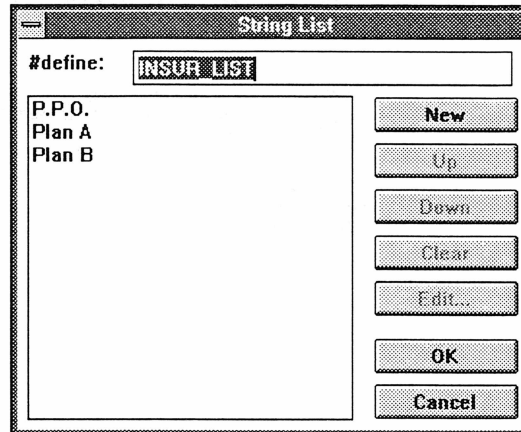


Introduction to XVT-Design++ Development

Slide 126

Course Example: Using the Strings Editor (Cont.)

- ❖ Select *New* 3 times and double click each item to change string



Introduction to XVT-Design++ Development

Slide 127

Controls: Additional Tags

❖ Event tags:

- ♦ CREATE - code that initializes the appearance of a control
- ♦ DESTROY - code that frees storage memory
- ♦ USER - code that responds to an application-defined event

❖ Special tags:

- ♦ CLASS DECL - place to add instance data to the class
- ♦ CONSTRUCTOR - code placed in the object's constructor
- ♦ DESTRUCTOR - code placed in the object's destructor

Introduction to XVT-Design++ Development

Slide 128



Custom Controls

- ❖ **An opportunity to define application-specific information displays and inputs:**
 - ♦ Sliders
 - ♦ Spreadsheets
 - ♦ Toolbars
- ❖ **Class and member functions defined by you in a file whose name will be referenced as the Class within the control's Attributes... dialog**
- ❖ **Can only be contained by a window**



Controls: Summary

- ❖ **Controls are always within a window or dialog**
- ❖ **Controls usually defined and created along with their container**
- ❖ **Various control types to support specific user interaction scenarios**
- ❖ **Key tag - ACTION events**
- ❖ **Custom controls are supported**

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

[illegible]

Section 5

Building an XVT-Design++ Application

 Introduction to XVT-Design++ Development

Slide 1

What This Section Covers

- ❖ **Windows with controls (only)**
- ❖ **Graphical output windows**
 - ◆ Viewport windows
 - ◆ Scaled-to-fit graphical output
- ❖ **Graphical output operations**
- ❖ **Direct manipulation of window contents**
- ❖ **Using Window Classes**
- ❖ **Using Child Windows**

 Introduction to XVT-Design++ Development

Slide 2

Course Example - Window Requirements

- ❖ **Display the database**
 - ♦ Will contain controls to manage database access
 - ♦ To be displayed as part of the application creation
- ❖ **Present the salary history**
 - ♦ To contain a graphic depiction of the salary history
 - ♦ The graphic should be interactive to allow changes

 Introduction to XVT-Design++ Development

Slide 3

Windows: GUI Concept

- ❖ Defines an application's work area on the desktop
- ❖ GUI container object for graphical output (text, images, shapes)
- ❖ Provides Views into application's data
- ❖ Client area contents are managed by the application code

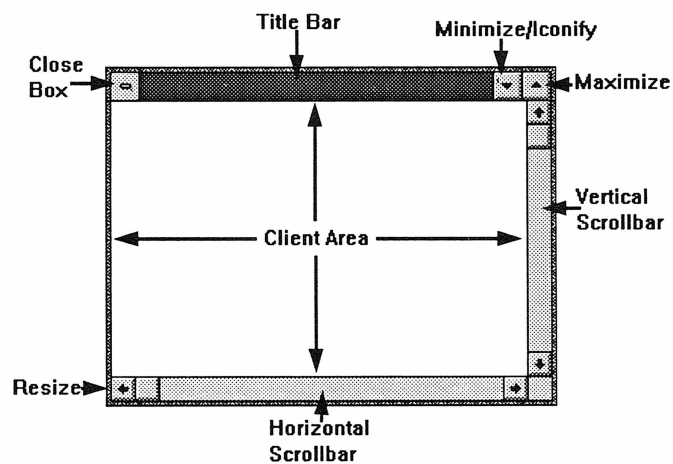
 Introduction to XVT-Design++ Development

Slide 4

Windows: GUI Concept (cont.)

- ❖ May logically or physically possess a menubar
- ❖ Optional window border decorations:
 - ♦ Title bar offers a grab handle for identifying and moving a window
 - ♦ Border scrollbars support the viewport model
 - ♦ Resize, iconize, and close border controls

Window Components



Windows vs Dialogs

❖ Windows

- ♦ Can be drawn in
- ♦ Get mouse events
- ♦ May contain custom controls and text edit objects
- ♦ Programmer can dynamically add and remove controls

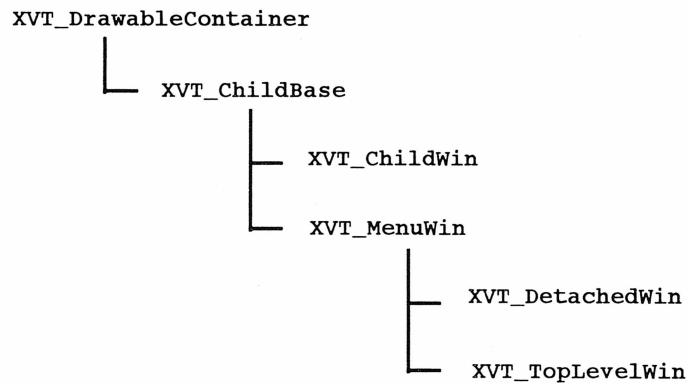
❖ Dialogs

- ♦ Provide a modality attribute
- ♦ Provide keyboard navigation
- ♦ May always appear in front (platform-specific)

Introduction to XVT-Design++ Development

Slide 7

Windows: Window Types



Introduction to XVT-Design++ Development

Slide 8

Windows: Creating Top-Level Windows

- ❖ **Derive from XVT_TopLevelWin or XVT_DetachedWin**
- ❖ **Window Styles**
 - ◆ Document Style - supports border decorations
 - ◆ Double Border
 - ◆ Single (Plain) Border
- ❖ **XVT-Design++ generates code to instantiate windows from a static resource definition (URL)**
- ❖ **Dynamically create windows by programmatically instantiating your own derived classes**

Introduction to XVT-Design++ Development

Slide 9

Windows: Window Attributes

- ❖ **Attributes for all top-level windows:**
 - ◆ Parent class
 - ◆ Type (document, plain, double border)
 - ◆ Initial size and location
 - ◆ Title
 - ◆ Visibility and "enabledness"
 - ◆ Menubar
- ❖ **Document-style window attributes**
 - ◆ Sizable
 - ◆ Close box
 - ◆ Horizontal and/or vertical border scrollbars
 - ◆ Iconizable / iconized / maximized

Introduction to XVT-Design++ Development

Slide 10

Course Example -Database Window Layout and Connections

❖ System requirements

- ♦ Window with controls to display database status
- ♦ Window's with push buttons to manage database access

❖ Window and controls layout and attributes

- ♦ Task menubar
- ♦ Control justification

❖ Connections

- ♦ Application CREATE event and Database window
- ♦ Push button ACTION events - stub out

Introduction to XVT-Design++ Development

Slide 11

Course Example: Creating the "Database Window"

The screenshot shows a window titled "Databases". Inside, it displays the following information:

- Databases: Employee Database
- Status: Opened
- # of Records: 5
- Description: This database holds all Employee Information.

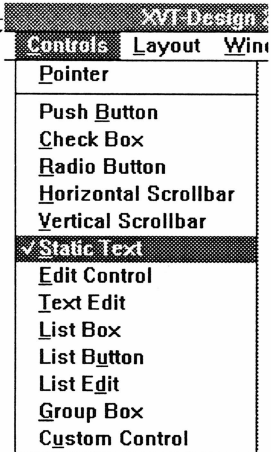
At the bottom of the window, there are three buttons: "Access", "Browse...", and "Disconnect".

Introduction to XVT-Design++ Development

Slide 12

Course Example: Add Controls to Databases Window

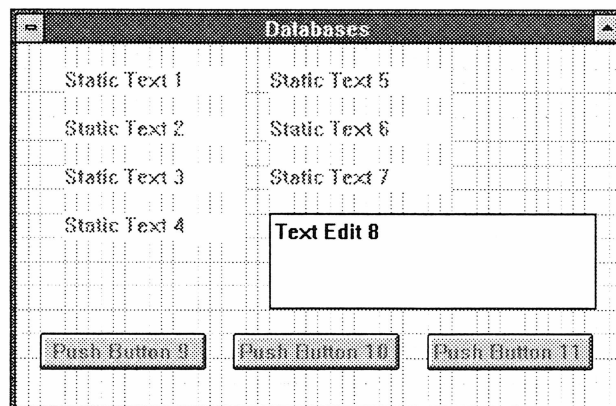
- ❖ Hold shift key for repeat placement
- ❖ Select *Pointer* to end repeat
- ❖ Add 7 static text control labels
- ❖ Add 1 Text Edit Object control
- ❖ Add 3 pushbutton Controls



Introduction to XVT-Design++ Development

Slide 13

Course Example: Databases Window with Controls



Introduction to XVT-Design++ Development

Slide 14

Course Example: Setting Control Attributes

- ❖ 4 static text controls will be labels (right justify)
 - ♦ Database:
 - ♦ Status:
 - ♦ # of Records:
 - ♦ Description:
- ❖ 3 static text controls will be empty labels
- ❖ Application will set titles for empty static text labels at runtime

Introduction to XVT-Design++ Development

Slide 15

Course Example: Setting Control Attributes

- ❖ 4 static text controls will be labels (right justify)
 - ♦ Database:
 - ♦ Status:
 - ♦ # of Records:
 - ♦ Description:
- ❖ 3 static text controls will be empty labels
- ❖ Application will set titles for empty static text labels at runtime

Introduction to XVT-Design++ Development

Slide 16

Course Example: Setting Control Attributes (Cont.)

❖ Set #define names for empty static text labels

- ♦ DBW_DATABASES_ST
- ♦ DBW_STATUS_ST
- ♦ DBW_NUMRECORDS_ST

❖ Set #define name for text edit object

<u>Title</u>	<u>#define</u>
	DBW_DESC_TE

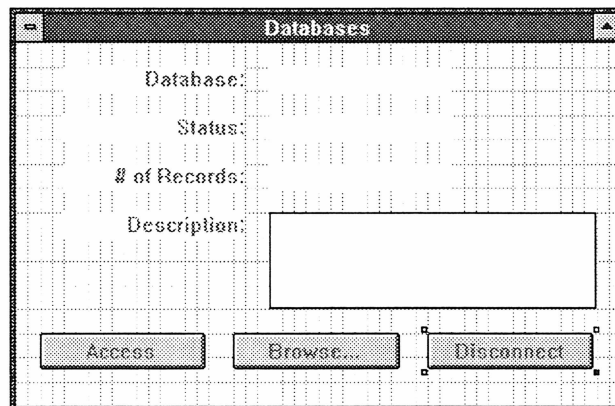
❖ Set #define names and titles for 3 pushbuttons

<u>Title</u>	<u>#define</u>
Access	DBW_ACCESS_PB
Browse...	DBW_BROWSE_PB
Disconnect	DBW_DISCONNECT_PB

Introduction to XVT-Design++ Development

Slide 17

Course Example: Databases Window After Changing Control Attributes

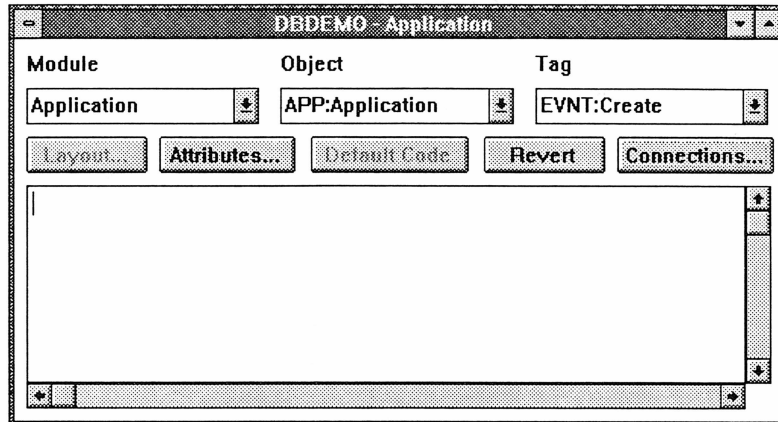


Introduction to XVT-Design++ Development

Slide 18

Course Example: Set Connections

❖ **Select *Connections...* from ACE**



Introduction to XVT-Design++ Development

Slide 19

Course Example: Setting Connections (cont.)

- ❖ **Want to set a connection to create the Databases window at application startup**
- ❖ **ACE settings**
 - ♦ Module=Application, Object=Application, Tag=Create
 - ♦ Select *Connections...* pushbutton
 - ♦ Means: during application creation, set a connection
- ❖ **Want to create the user defined Databases window object**

Introduction to XVT-Design++ Development

Slide 20

Course Example: Setting Connections (cont.)

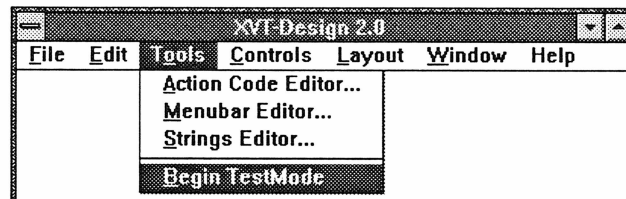
- ❖ Want to set a connection to create the Databases window at application startup
- ❖ ACE settings
 - ♦ Module=Application, Object=Application, Tag=Create
 - ♦ Select *Connections...* pushbutton
 - ♦ Means: during application creation, set a connection
- ❖ Want to create the user defined Database window object

Introduction to XVT-Design++ Development

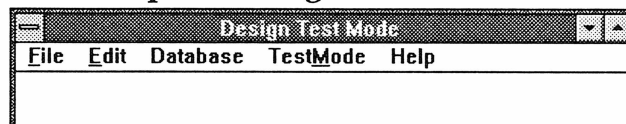
Slide 21

Course Example: Running TestMode

- ❖ Select from *Tools* menu



- ❖ Class example running TestMode



Introduction to XVT-Design++ Development

Slide 22

Course Example: Creating the "Databases Window" (Cont.)

- ❖ Select "New Window" from the Windows menu
 - ♦ This places a newly created window on the screen
- ❖ Set the window title and the #define name
 - ♦ Double Click on window, or select Window 101 from Modules list and select Attributes
 - ♦ Set Title field to "Databases"
 - ♦ Set #define field to "DB_WIN"
 - ♦ Size to taste using resize handles

Introduction to XVT-Design++ Development

Slide 23

Course Example: Window Attributes Dialog

Change Window Attributes

Title: Class:

#define:

Border: ☒ Document ☐ Double ☐ Plain

☐ Invisible ☐ Iconizable ☐ Size Box

☐ Disabled ☐ Iconized ☐ Vert Scrollbar

☐ Close Box ☐ Maximized ☐ Horz Scrollbar

X: Width:

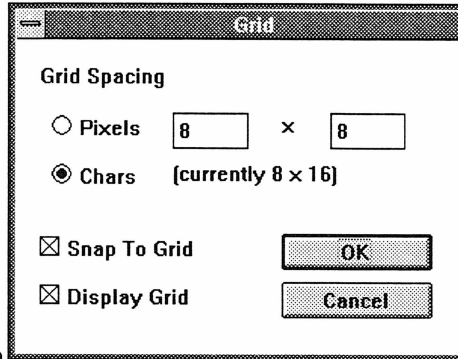
Y: Height:

Introduction to XVT-Design++ Development

Slide 24

Class Example: Setting Grid for Databases Window

- ❖ May port to XVT/CH, so setting character grid early on will help layout on other systems
- ❖ Choose Grid from Layout menu
 - ♦ Select "Chars"
 - ♦ Select "Snap to Grid"
 - ♦ Select "Display Grid"



Introduction to XVT-Design++ D

Windows with Controls: Handling Events

- ❖ As with a dialog, handle events:
 - ♦ CREATE
 - ♦ CLOSE
 - ♦ DESTROY
 - ♦ Controls get ACTION events
- ❖ New, UPDATE, event to handle

Introduction to XVT-Design++ Development

Slide 26

*Windows with Controls: Handling **CREATE** Events*

- ❖ The first event received by the window
- ❖ Notification that the window has been created
- ❖ No arguments are passed
- ❖ Control objects instantiated in code generated by XVT-Design++
- ❖ You write code to initialize state of controls
 - ♦ Check boxes, radio buttons, list selections, edit fields

Introduction to XVT-Design++ Development

Slide 27

*Windows with Controls: Handling **DESTROY** Events*

- ❖ The last event received by a window
- ❖ Notification that the window has been closed
- ❖ No arguments are passed
- ❖ Control objects automatically deleted
- ❖ You free any memory or resources used by your code

Introduction to XVT-Design++ Development

Slide 28

Windows with Controls: Handling CLOSE Events

- ❖ Request by user to close the window
- ❖ No arguments are passed
- ❖ Default code generated by XVT-Design++ calls `XVT_DrawableContainer::Close()`
- ❖ You decide whether `Close()` should check for work not saved
 - ◆ Most native applications do not

Introduction to XVT-Design++ Development

Slide 29

Windows with Controls: Handling UPDATE Events

- ❖ Request to update the graphical contents of a window's client area
 - ◆ If controls only, no graphical output
 - ◆ Controls are drawn (and updated) by XVT++ or native WS
 - ◆ Default code is sufficient
- ❖ Default code generated by XVT-Design++ calls `XVT_DrawableContainer::Clear()`

Introduction to XVT-Design++ Development

Slide 30

Window Special Tags

- ❖ **CLASS DECL** - place to declare data and functions
- ❖ **CLASS HEADER** - place to add #include's to .h file
- ❖ **TOP** - place to add declarations, constants, etc.
- ❖ **BOTTOM** - place to define member functions
- ❖ **CONSTRUCTOR** - place to add code to constructor
- ❖ **DESTRUCTOR** - place to add code to destructor

Introduction to XVT-Design++ Development

Slide 31

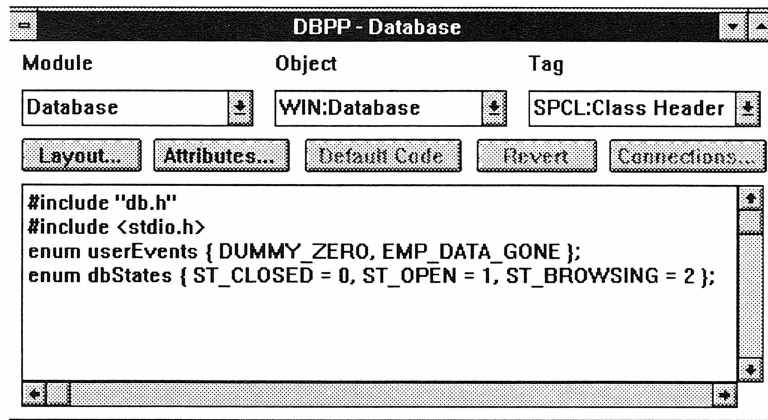
Course Example - Window Action Code

- ❖ **Action code associated with special tags**
 - ♦ **CLASS HEADER** - #include's and enums
 - ♦ **CLASS DECL** - instance data and member functions
 - ♦ **CONSTRUCTOR** - store's database identifier

Introduction to XVT-Design++ Development

Slide 32

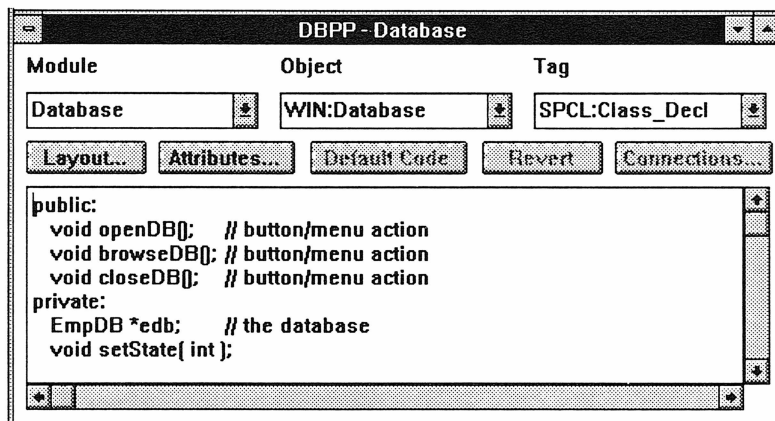
Database Window Action Code: *Class_Header tag*



Introduction to XVT-Design++ Development

Slide 33

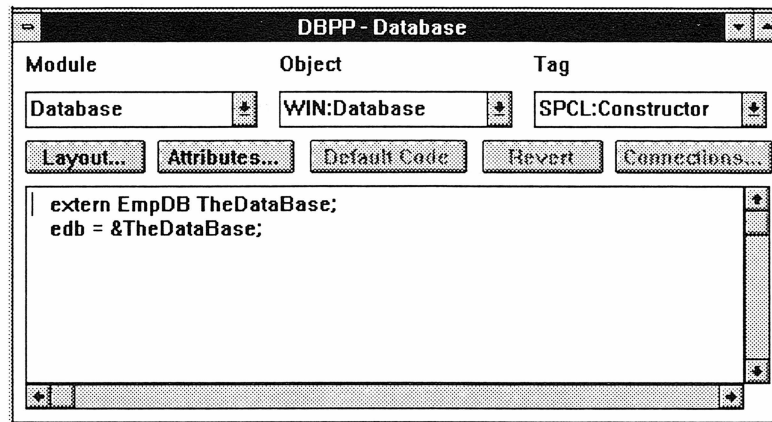
Database Window Action Code: *Class_Decl tag*



Introduction to XVT-Design++ Development

Slide 34

Database Window Action Code: Constructor tag



Introduction to XVT-Design++ Development

Slide 35

Windows: Member Function Summary

- ❖ `BOOLEAN Init(long rid)`
- ❖ `BOOLEAN Init(WIN_TYPE wtype, XVT_Rct boundary, ...)`
- ❖ `virtual WIN_TYPE GetType() const`
- ❖ `virtual XVT_Base* GetParent() const`
- ❖ `XVT_Menu* GetMenu()`
- ❖ `XVT_Menu* SetMenu(XVT_Menu* menu)`
- ❖ `BOOLEAN GetTitle(char* buffer, unsigned long* len) const`
- ❖ `void SetTitle(const char* str)`
- ❖ `void SetDocTitle(const char* str)`
- ❖ `void Clear(XVT_Color color)`
- ❖ `void Close()`

Introduction to XVT-Design++ Development

Slide 36

Windows: Member Function Summary

- ❖ `void MakeFront()`
- ❖ `BOOLEAN GetEnabledState() const`
- ❖ `void SetEnabledState(BOOLEAN state)`
- ❖ `BOOLEAN GetVisibleState() const`
- ❖ `void SetVisibleState(BOOLEAN state)`
- ❖ `void SetEventMask(EVENT_MASK mask)`
- ❖ `EVENT_MASK GetEventMask() const`
- ❖ `void SetInnerRect(XVT_Rct boundary)`
- ❖ `virtual XVT_Rct GetInnerRect() const`
- ❖ `virtual XVT_Rct GetOuterRect() const`

Introduction to XVT-Design++ Development

Slide 37

Windows with Controls: Member Function Summary

- ❖ `virtual void e_create()`
- ❖ `virtual void e_close()`
- ❖ `virtual void e_destroy()`
- ❖ `virtual void e_update(XVT_Rct boundary)`
- ❖ `XVT_Control* GetCtl(long cid)`
- ❖ `long GetCtlCount() const`
- ❖ `XVT_Control* GetFirstCtl()`
- ❖ `XVT_Control* GetNextCtl()`
- ❖ `XVT_TextEdit* GetTextEdit(long id)`
- ❖ `XVT_TextEdit* GetActiveTextEdit()`

Introduction to XVT-Design++ Development

Slide 38

Graphical Output Windows

- ❖ **Coordinate systems**
- ❖ **Types of graphical output**
- ❖ **Two styles of graphical output windows**
 - ♦ Viewport
 - ♦ Scaled-to-fit
- ❖ **New event types to handle**
 - ♦ SIZE, UPDATE, FONT, and SCROLL
 - ♦ No consideration of input events (MOUSE_*) yet
- ❖ **Portability considerations**

Introduction to XVT-Design++ Development

Slide 39

Graphical Output: Coordinate Systems

- ❖ **Drawing coordinates specified in units of pixels**
 - ♦ Origin at upper left corner
- ❖ **Global physical coordinate system defined by XVT_ScreenWin**
- ❖ **Every window has its own physical coordinate system**
- ❖ **Your application will use its own logical coordinate system**
 - ♦ You define coordinate transformations from logical to physical
- ❖ **XVT_GlobalAPI::TranslatePoints() translates from one window's coordinates to another window's**

Introduction to XVT-Design++ Development

Slide 40

Graphical Output: Types of Output

- ❖ **Text**
 - ♦ Variety of font families, font styles, point sizes
- ❖ **Lines**
 - ♦ Line style, color, thickness, arrow heads
 - ♦ Polylines and arcs
- ❖ **Closed shapes**
 - ♦ Border style, fill style
 - ♦ Regular shapes - ovals, rectangles,
 - ♦ Irregular shapes - polygons, pie shapes
- ❖ **Icons**
- ❖ **Pictures - collections of output operations**

Introduction to XVT-Design++ Development

Slide 41

Graphical Output Windows - Viewport Model

- ❖ **Scrollbars allow moving viewport on larger virtual page**
 - ♦ Examples - word processor and layout programs
- ❖ **Application must maintain a concept of viewport origin**
 - ♦ Client area upper-left versus virtual page upper-left
 - ♦ Example - character in upper left corner is actually line 217, column 43 of the document
- ❖ **You calculate how much is visible versus the size of the virtual page**

Introduction to XVT-Design++ Development

Slide 42

Graphical Output Windows - Scaled-to-Fit Model

- ❖ **Drawing, and possibly text, is scaled to fit in the client area**
 - ♦ Charts and graphs
- ❖ **Coordinate transformations required**
 - ♦ The mapping changes each time the window size changes

Introduction to XVT-Design++ Development

Slide 43

Graphical Output Windows: Handling *SIZE* Events

- ❖ **Notification of the size of a window's client area**
 - ♦ Argument provides rectangle with new height and width
- ❖ **Size notification is delivered to the window:**
 - ♦ When the window is created - initial size
 - ♦ Whenever the user resizes the window
 - ♦ When border scrollbars appear or disappear
- ❖ **No default code generated by XVT-Design++**
- ❖ **You add code to:**
 - ♦ Viewport model: Re-calculate and set scrollbar proportions
 - ♦ Scaled-to-fit model: Change your coordinate mapping, possibly recalculate point sizes for fonts, force a redraw of the entire window using `XVT_BaseDrawProto::Invalidate()`

Introduction to XVT-Design++ Development

Slide 44

Graphical Output Windows: Handling UPDATE Events

- ❖ Notification of “damage” to a window’s client area
 - ♦ Argument provides rectangle bounding the damaged region
- ❖ Update notification is delivered to the window:
 - ♦ When the window is created - draw initial contents
 - ♦ Whenever the window’s contents have been damaged - physically or logically
 - ♦ You can induce UPDATE events by calling `Invalidate()` when the Model changes
- ❖ Be prepared to redraw any window’s contents at any time

Introduction to XVT-Design++ Development

Slide 45

Graphical Output Windows: Handling UPDATE Events (cont.)

- ❖ Default XVT-Design++ generated code calls `Clear()`
 - ♦ Erases damaged area
- ❖ You add code to draw your graphical output
 - ♦ Viewport model: Optimize your drawing algorithm to only draw in viewport
 - ♦ Temporary clipping region may be in effect
 - ♦ Additional optimization possible using `XVT_BaseDrawProto::NeedsUpdate()`
- ❖ Only do drawing in response to UPDATE events
 - ♦ Many XVT function calls are illegal during update processing due to constraints of native platforms
 - ♦ Illegal calls may be suppressed via `SetAttrValue()` and `ATTR_SUPPRESS_UPDATE_CHECK`

Introduction to XVT-Design++ Development

Slide 46

Graphical Output Windows: Handling FONT Events

- ❖ **Request to change the font used to draw text in a window's client area**
 - ♦ Arguments provide portable font data structure and indication of which font characteristic changed
- ❖ **Default code generated by XVT-Design++:**
 - ♦ Installs new font into the drawing protocol with `XVT_BaseDrawProto::SetFont()`
 - ♦ Updates the font menu with `XVT_MenuWin::SetFontMenu()`
 - ♦ Forces an UPDATE to redraw using the new font with `XVT_DrawableContainer::Invalidate()`
- ❖ **Window uses single font or multiple fonts?**
 - ♦ Multiple fonts require more complex data structures

Introduction to XVT-Design++ Development

Slide 47

Viewport Windows: Handling Window Scrollbar Events

- ❖ **Notification that window border scrollbar was operated by user**
 - ♦ Different member functions for vertical and horizontal scrollbar
- ❖ **Argument indicates which part of the scrollbar was operated**
 - ♦ Line Up, Line Down, Page Up, Page Down, Thumb, Thumb Track
 - ♦ Interpretation of "line" and "page" is application-dependent

Introduction to XVT-Design++ Development

Slide 48

Viewport Windows: Handling Window Scrollbar Events (cont.)

- ❖ Thumb manipulation also provides the new position within the scrollbar range
- ❖ You write code to scroll window contents and position the thumb using:
 - ♦ `XVT_DrawableContainer::Scroll()`
 - ♦ `XVT_ChildBase::SetScrollPosition()`

Introduction to XVT-Design++ Development

Slide 49

Graphical Output Windows: Portability Considerations

- ❖ Resolution (number of pixels per inch) varies across screens
- ❖ Aspect ratio - pixels aren't necessarily square
 - ♦ Take into account when drawing shapes
- ❖ Attributes provide resolution and aspect ratio information
 - ♦ `XVT_GlobalAPI::GetAttrValue()`
 - ♦ Attributes - `ATTR_SCREEN_HRES`, `ATTR_SCREEN_VRES`, `ATTR_SCREEN_HEIGHT`, `ATTR_SCREEN_WIDTH`
 - ♦ Available for screen windows and print windows

Introduction to XVT-Design++ Development

Slide 50

Graphical Output Windows: Member Function Summary

- ❖ `virtual void e_font(XVT_Font font, FONT_PART part)`
- ❖ `virtual void e_size(XVT_Rct boundary)`
- ❖ `virtual void e_update(XVT_Rct boundary)`
- ❖ `virtual void e_hscroll(SCROLL_CONTROL activity, long pos)`
- ❖ `virtual void e_vscroll(SCROLL_CONTROL activity, long pos)`
- ❖ `void SetFontMenu(XVT_Font font)`
- ❖ `void Invalidate(XVT_Rct boundary)`
- ❖ `void Clear(XVT_Color color)`

Introduction to XVT-Design++ Development

Slide 51

Graphical Output Windows: Member Function Summary

- ❖ `void Scroll(XVT_Rct boundary, long dh, long dv)`
- ❖ `long GetScrollPosition(SCROLL_TYPE scroll_type) const`
- ❖ `void SetScrollPosition(SCROLL_TYPE scroll_type, long position)`
- ❖ `long GetScrollProportion(SCROLL_TYPE scroll_type) const`
- ❖ `void SetScrollProportion(SCROLL_TYPE scroll_type, long proportion)`
- ❖ `long GetScrollRange(SCROLL_TYPE scroll_type, long* min, long* max) const`
- ❖ `void SetScrollRange(SCROLL_TYPE scroll_type, long min, long max, long pos)`

Introduction to XVT-Design++ Development

Slide 52

Course Example - Graphics Requirements

- ❖ Where are graphics needed?
 - ♦ To display salary history information
- ❖ What kinds of graphics?
 - ♦ Lines or filled rectangles to graph or chart data
 - ♦ Lines for axes
 - ♦ Text labels

Introduction to XVT-Design++ Development

Slide 53

Course Example: Graphical Window

Change Window Attributes

Title: Class:

#define:

Border: ☒ Document ☐ Double ☐ Plain

☐ Invisible ☐ Iconizable ☒ Size Box

☐ Disabled ☐ Iconized ☐ Vert Scrollbar

☒ Close Box ☐ Maximized ☐ Horz Scrollbar

X: Width:

Y: Height:

Introduction to XVT-Design++ Development

Slide 54

Graphical Output: GUI Concepts

- ❖ Everything displayed in a window is graphical (except for controls)
- ❖ Drawing primitives used to display lines, shapes and text
- ❖ Graphical attributes determine graphics appearance
 - ♦ Line styles, fill colors, drawing modes, font styles, etc.
- ❖ Graphical output clipped to window's client area
 - ♦ Additional clipping rectangle may be set
- ❖ Application must remember window contents and redraw on each UPDATE

Introduction to XVT-Design++ Development

Slide 55

Graphical Output: Drawing Primitives

- ❖ Graphics shapes are drawn using the `XVT_BaseDrawProto` member functions:

<code>DrawALine()</code>	<code>DrawArc()</code>
<code>DrawIcon()</code>	<code>DrawLine()</code>
<code>DrawOval()</code>	<code>DrawPie()</code>
<code>DrawPolygon()</code>	<code>DrawPolyline()</code>
<code>DrawRect()</code>	<code>DrawRoundedRect()</code>

- ❖ **Reminder: Graphics are always drawn using the window's current drawing attributes**

Introduction to XVT-Design++ Development

Slide 56

Graphical Output: Drawing Primitives (cont.)

- ❖ Text is drawn using the member function
`XVT_BaseDrawProto::DrawText()`
 - ♦ An `XVT_Pnt` argument defines the baseline on which the text will be drawn
- ❖ A collection of graphical primitives may be re-drawn using the member function
`XVT_BaseDrawProto::DrawPicture()`

Graphical Output: Points and Rectangles

- ❖ Drawing primitives take `XVT_Pnt` or `XVT_Rct` objects as arguments
- ❖ Many styles of constructors defined
 - ♦ `XVT_Pnt::XVT_Pnt(short x=0, short y=0)`
 - ♦ `XVT_Rct::XVT_Rct(XVT_Pnt top_left, XVT_Pnt bottom_right)`
 - ♦ `XVT_Rct::XVT_Rct(long top, long left, long bottom, long right), etc.`
- ❖ Many operators defined for combining points, rectangles and offsets
- ❖ Member functions provide access to coordinates
 - ♦ `XVT_Pnt::GetX(), XVT_Pnt::GetY()`
 - ♦ `XVT_Rct::GetTopLeft(), XVT_Rct::GetBottomRight(), etc.`

Graphical Output: Points and Rectangles (cont.)

❖ Other operations on points and rectangles

```
XVT_Rct::IsEmpty()  
XVT_Rct::Contains( XVT_Pnt point )  
XVT_Rct::Contains( XVT_Rct rectangle )  
XVT_Rct::Intersect( XVT_Rct& boundary,  
    XVT_Rct *intersection )
```

Graphical Output: Drawing Attributes

❖ Window classes derived from

`XVT_DrawableContainer` inherit a
`XVT_BaseDrawProto* DrawProtocol`

❖ DrawProtocol maintains a set of drawing tools for the window

- ♦ Pens - used to draw lines and the borders of closed shapes
- ♦ Brushes - used to fill the interior of closed shapes
- ♦ Drawing Modes - determines how drawn pixels are combined with pixels already on the screen
- ♦ Fonts - consist of a family, style, and size
- ♦ Color - of the pen, brush, foreground and background

❖ DrawProtocol maintains a clipping region

- ♦ Drawing operations may be clipped to a sub-region of the client area

Graphical Output: Pens

- ❖ A pen is used to draw lines and the borders of closed shapes
- ❖ Represented by the class `XVT_Pen`
- ❖ Pen attributes
 - ♦ Width - in pixels
 - ♦ Pattern - Solid, Hollow or Rubber
 - ♦ Style - Solid, Dotted or Dashed
 - ♦ Color
- ❖ Default attributes are set for you
 - ♦ 1-pixel wide solid black pen
- ❖ Change the pen before drawing by calling `XVT_BaseDrawProto::SetPen()`

 Introduction to XVT-Design++ Development

Slide 61

Graphical Output: Brushes

- ❖ A brush is used to fill the interior of closed shapes
- ❖ Represented by the class `XVT_Brush`
- ❖ Brush attributes
 - ♦ Pattern - Solid, Hollow, variety of hatch patterns
 - ♦ Color of solid area or of hatch lines
- ❖ Default attributes
 - ♦ Solid white brush
- ❖ Change the brush before drawing by calling `XVT_BaseDrawProto::SetBrush()`

 Introduction to XVT-Design++ Development

Slide 62

Graphical Output: Drawing Modes


- ❖ Drawing mode determines how drawn pixels combine with pixels already on the screen
 - ♦ M_COPY, M_OR, M_XOR, etc...
- ❖ M_COPY is the default mode
- ❖ M_XOR combines pixels using these rules:
 - ♦ Drawing a shape once with the M_XOR mode will allow you to see the shape if at all possible
 - ♦ Drawing the same shape again with the M_XOR mode is guaranteed to leave the window as it was before the first drawing operation

 Introduction to XVT-Design++ Development

Slide 63

Graphical Output: Fonts

- ❖ DrawText () uses the current font to render the text segment given to it
- ❖ Represented by the class XVT_Font
- ❖ Font attributes
 - ♦ Family - FF_SYSTEM, FF_FIXED, FF_TIMES, FF_HELVETICA
 - ♦ Style - FS_BOLD, FS_ITALIC
 - ♦ Size - short integer point size
- ❖ Default attributes are set for you, based on the platform's recommended system font
- ❖ Change the font before drawing by calling XVT_BaseDrawProto::SetFont ()

 Introduction to XVT-Design++ Development

Slide 64

Graphical Output: Font Mapper

- ❖ You request a font object with the constructor
`XVT_Font::XVT_Font(long family, long style, short size)`
- ❖ That exact font object may not be available on this platform
- ❖ A font object with the closest attribute match is provided

Graphical Output: Font Metrics

- ❖ **Font Metrics**
 - ♦ Ascent - distance from baseline to top of tallest character
 - ♦ Descent - distance from baseline to bottom of descenders
 - ♦ Leading - recommended interline spacing
- ❖ **Attributes differ for each font**
- ❖ **Query the current attributes with**
`XVT_BaseDrawProto::GetFontMetrics()`
- ❖ **Use font metrics for calculating line spacing in text output windows**

Font Metrics

Ascent { Ker ky Baseline
Descent {
Leading {
Lo ana Baseline

Introduction to XVT-Design++ Development

Slide 67

Graphical Output: Color

- ❖ Colors are specified for the pen, brush, foreground and background
 - ♦ Previous slides discussed pen and brush colors
 - ♦ Foreground color is used for drawn text
 - ♦ Background color is used for spaces around text and for the spaces between the brush pattern lines
- ❖ Represented by the class `XVT_Color`
- ❖ 11 Predefined colors for the constructor

`XVT_Color(COLOR)`

<code>COLOR_RED</code>	<code>COLOR_GREEN</code>	<code>COLOR_BLUE</code>
<code>COLOR_CYAN</code>	<code>COLOR_MAGENTA</code>	<code>COLOR_YELLOW</code>
<code>COLOR_BLACK</code>	<code>COLOR_DKGRAY</code>	<code>COLOR_GRAY</code>
<code>COLOR_LTGRAY</code>	<code>COLOR_WHITE</code>	

Introduction to XVT-Design++ Development

Slide 68

Graphical Output: Color (cont.)

- ❖ XVT++ uses a 24 bit RGB color model
- ❖ To make your own colors, use the constructor

```
XVT_Color( unsigned short red=0,
            unsigned short green=0,
            unsigned short blue=0 )
```

Graphical Output: Clipping Rectangle

- ❖ UPDATE event maintains a notion of the damaged region
 - ♦ Drawing operations during UPDATE processing will be clipped to that region
- ❖ An additional clipping rectangle can be set with `XVT_BaseDrawProto::SetClip()`
 - ♦ Drawing during UPDATE processing will be clipped to the intersection of the two rectangles
 - ♦ Drawing operations at other times are clipped to DrawProtocol's clipping rectangle only
- ❖ Clipping can be turned on and off with `XVT_BaseDrawProto::SetClipState()`

Graphical Output: XVT++ Pictures

- ❖ **A picture encapsulates a series of drawing operations into a single object**
 - ♦ Operations include tool changes, drawing shapes, and text output
- ❖ **Represented by the class XVT_Picture**
- ❖ **Pictures can be**
 - ♦ Displayed in a window
 - ♦ Put on the clipboard
 - ♦ Saved to or retrieved from a file

Introduction to XVT-Design++ Development

Slide 71

Adding Input and Direct Manipulation

- ❖ **New event types to handle**
 - ♦ FOCUS, CHAR and MOUSE_* events
- ❖ **Additional support for text input**
 - ♦ Carets show insertion point
- ❖ **Mouse-related abstractions**
 - ♦ Cursors
 - ♦ Trapping the mouse
- ❖ **Drawing outside context of the UPDATE event**

Introduction to XVT-Design++ Development

Slide 72

Direct Manipulation Windows: Handling FOCUS Events

- ❖ Notification that the window has gained or lost keyboard focus
 - ♦ Argument is TRUE if the window is gaining focus and FALSE if it is losing focus
 - ♦ Only one container may have keyboard focus on the entire desktop
- ❖ Typical response to FOCUS events:
 - ♦ Check the clipboard for data availability (if needed)
 - ♦ Show graphical selections to be either highlighted or normal
- ❖ Activation events (argument TRUE) are guaranteed to be paired with deactivation events

Introduction to XVT-Design++ Development

Slide 73

Direct Manipulation Windows: Handling CHAR Events

- ❖ Notification that the user hit a key on the keyboard
 - ♦ Containers can only receive character events if they have focus
 - ♦ Keyboard input may be intercepted by controls
 - ♦ Menu accelerators do *not* come through as keyboard events
 - ♦ Function keys are represented by virtual key codes
- ❖ Arguments define the character, or key-chord, typed
- ❖ You write code to respond to the key press
 - ♦ Update internal data structures
 - ♦ May also immediately draw the character on the display

Introduction to XVT-Design++ Development

Slide 74

*Direct Manipulation Windows: Handling MOUSE_ * Events*

- ❖ Notification that a mouse button was pressed, released, or that the mouse was moved
- ❖ Arguments supply:
 - ♦ Mouse button number
 - ♦ The mouse position
 - ♦ Keyboard modifiers (shift and control key flags)
- ❖ Mouse buttons 0 and 1 supported across all platforms

Introduction to XVT-Design++ Development

Slide 75

*Direct Manipulation Windows: Handling MOUSE_ * Events (cont.)*

- ❖ **MOUSE_DOWN** - the user pressed a mouse button
- ❖ **MOUSE_UP** - the user released the button
 - ♦ A click is two events: **MOUSE_DOWN** followed by **MOUSE_UP**
 - ♦ A **MOUSE_UP** occurs only if the user releases the button over the same window
- ❖ **MOUSE_DBL** - a button was pressed twice rapidly
 - ♦ Event stream: **MOUSE_DOWN**, **MOUSE_UP**, **MOUSE_DBL**, **MOUSE_UP**
- ❖ **MOUSE_MOVE** - the user moved the mouse, regardless of whether a button is down or up

Introduction to XVT-Design++ Development

Slide 76

*Direct Manipulation Windows: Handling MOUSE_ * Events (cont.)*

❖ MOUSE_DOWN

- ♦ Save location of click
- ♦ Perform hit test to locate selection point or object
- ♦ Set application flag for possible drag operation

❖ MOUSE_MOVE

- ♦ Extend selection
- ♦ Drag object
- ♦ Auto-scroll window contents

Introduction to XVT-Design++ Development

Slide 77

*Direct Manipulation Windows: Handling MOUSE_ * Events (cont.)*

❖ MOUSE_UP

- ♦ End selection
- ♦ Activate Cut, Copy, Clear menu items
- ♦ Perform hit test to determine drop location
- ♦ Clear application flag for possible drag operation

❖ MOUSE_DBL

- ♦ Extend the semantic of a single click (select entire word, select object, open, etc)

Introduction to XVT-Design++ Development

Slide 78

Direct Manipulation Windows: Carets

- ❖ Text input requires a visual indication of where the next character will be placed
- ❖ You write code to
 - ♦ Make the caret visible or invisible with `XVT_ChildBase::SetCaretState()`
 - ♦ Echo each character after it is typed
 - ♦ Reposition the caret with `XVT_ChildBase::SetCaretPos()`
- ❖ You may optionally want to change the dimensions of the caret
- ❖ `XVT_TextEdit` objects handle this for you

Introduction to XVT-Design++ Development

Slide 79

Direct Manipulation Windows: Cursors

- ❖ Mouse input requires a visual indication of where the mouse is currently pointing
- ❖ XVT++ defines several cursor styles
 - ♦ Arrow, I-Beam, cross-hair, plus sign, waiting
 - ♦ Arbitrary user cursors can also be defined
- ❖ Each XVT++ window can have its own cursor
 - ♦ Default cursor is an arrow cursor
- ❖ You write code to change the default cursor style for any window with `XVT_ChildBase::SetCursor()`
 - ♦ Each window remembers it's own cursor setting

Introduction to XVT-Design++ Development

Slide 80

Direct Manipulation Windows: Trapping the Mouse

- ❖ **Mouse trapping logically sends mouse events to a window, even if the mouse cursor leaves the window**
 - ♦ Make sure you see a `MOUSE_UP`
 - ♦ Required for dragging operations
- ❖ **While trapped, a continuous stream of `MOUSE_MOVE` events is generated**
 - ♦ Useful for implementing autoscrolling of text and graphics
- ❖ **Cursor does not change while mouse is trapped**

Introduction to XVT-Design++ Development

Slide 81

Direct Manipulation Windows: Drawing Without an UPDATE Event

- ❖ **In a few cases, waiting for an `UPDATE` event is too slow**
 - ♦ Character input display
 - ♦ Mouse selections
 - ♦ Dragging and rubberbanding
 - ♦ Autoscrolling
 - ♦ Animation
- ❖ **Draw immediately, but make sure data structures are consistent so `UPDATE` events can still be handled**

Introduction to XVT-Design++ Development

Slide 82

Direct Manipulation Windows: Member Function Summary

- ❖ `virtual void e_focus(BOOLEAN active)`
- ❖ `virtual void e_char(short chr, BOOLEAN shift, BOOLEAN control)`
- ❖ `virtual void e_mouse_dbl(XVT_Pnt point, BOOLEAN shift, BOOLEAN control, short button)`
- ❖ `virtual void e_mouse_down(XVT_Pnt point, BOOLEAN shift, BOOLEAN control, short button)`
- ❖ `virtual void e_mouse_move(XVT_Pnt point, BOOLEAN shift, BOOLEAN control, short button)`
- ❖ `virtual void e_mouse_up(XVT_Pnt point, BOOLEAN shift, BOOLEAN control, short button)`

Introduction to XVT-Design++ Development

Slide 83

Direct Manipulation Windows: Member Function Summary

- ❖ `XVT_Pnt GetCaretPos() const`
- ❖ `void SetCaretPos(XVT_Pnt point)`
- ❖ `BOOLEAN GetCaretState() const`
- ❖ `void SetCaretState(BOOLEAN state)`
- ❖ `void SetCaretDimensions(XVT_Pnt vector)`
- ❖ `void SetCursor(CURSOR cursor)`
- ❖ `void TrapMouse()`
- ❖ `void ReleaseMouse()`


Introduction to XVT-Design++ Development

Slide 84



Using Window Classes

- ❖ **Application-specific windows**
 - ♦ A specialization of an XVT++ window class
- ❖ **You define the class and member functions in a separate file**
 - ♦ Best to use generated code as the starting point
- ❖ **Window class name in the Attributes... dialog is used to derive the file name**



Introduction to XVT-Design++ Development

Slide 85



Using Window Classes (cont.)

- ❖ **The Course Example uses a window class to display bar charts**
- ❖ **It wasn't necessary to use a window class, but we wanted to demonstrate the concept**



Introduction to XVT-Design++ Development

Slide 86

Windows: Additional Event Tags

- ❖ **TIMER** - notification that the window's timer expired, the `XVT_Timer` is an argument
- ❖ **USER** - used to implement your own protocols between different GUI objects

Introduction to XVT-Design++ Development

Slide 87

Creating Child Windows

- ❖ **Derive from `XVT_ChildWin`**
- ❖ **Window Styles**
 - ♦ Single (Plain) Border - may have scrollbars
 - ♦ No Border
- ❖ **XVT-Design++ generates code to instantiate custom controls**
- ❖ **Dynamically create child windows by entering action code to instantiate your own derived classes**

Introduction to XVT-Design++ Development

Slide 88

Child Windows: Member Function Summary

- ❖ `virtual XVT_Base* GetParent() const`
- ❖ `XVT_ChildBase* GetFirstWin()`
- ❖ `XVT_ChildBase* GetNextWin()`
- ❖ `long GetWinCount()`

Windows: Summary

- ❖ There are top-level and child windows
- ❖ Windows logically or physically possess a menubar
- ❖ UPDATE events are application critical
- ❖ Window use
 - ♦ Controls only, graphical output, or direct manipulation
 - ♦ Translates into different sets of events being relevant
- ❖ Window classing can be a useful extension to XVT-Design++

Graphics: Summary

- ❖ Window contents are drawn (with the exception of controls)
- ❖ The `XVT_BaseDrawProto` member functions provide the graphics primitives
- ❖ Graphics attributes include pens, brushes, drawing modes, colors and fonts
- ❖ Both global and local coordinate systems are used
- ❖ Pixel resolution and aspect ratio are screen dependent

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

[illegible]

Section 6

Completing an XVT-Design++ Application

Introduction to XVT-Design++ Development

Slide 1

What This Section Covers

- ❖ Additional Event Details
- ❖ Text Edit Objects
- ❖ Providing Clipboard Access
- ❖ Printing
- ❖ File Handling
- ❖ Error Handling and Debugging
- ❖ Memory Management

Introduction to XVT-Design++ Development

Slide 2

What This Section Covers (cont.)

- ❖ **Help System**
- ❖ **System Attribute Table**
- ❖ **Writing Non-Portable Code**

- ❖ **Application Generation**
- ❖ **Application Building and Running**

- ❖ **Commencement**

Introduction to XVT-Design++ Development

Slide 3

Additional Event Details: Handling USER Events

- ❖ **For message passing from one GUI object to another**
- ❖ **Invoke the `e_user(long id, void *data)` member function of the object you want to send a message to**
 - ♦ Distinguish different message types by `long id`
 - ♦ Provide arbitrary message data with `void *data`

Introduction to XVT-Design++ Development

Slide 4

*Additional Event Details: Handling **TIMER** Events*

- ❖ **Notification that an interval timer for a container has gone off**
 - ♦ Timers may be associated with all container types
- ❖ **Timers are represented by the class `XVT_Timer`**
 - ♦ Constructor takes container object and interval in milliseconds
- ❖ **Timers are created with the `new` operator and cancelled with the `delete` operator**

Introduction to XVT-Design++ Development

Slide 5

*Additional Event Details: Handling **TIMER** Events (cont.)*

- ❖ **XVT++ guarantees at least one timer will be available**
 - ♦ The maximum number available to an application on a platform is available from System Attributes
- ❖ **Minimum timer intervals are platform-specific**
 - ♦ 50 milliseconds is cross-platform minimum
- ❖ **Applications of interval timers**
 - ♦ Connection or window timeouts
 - ♦ Input device polling
 - ♦ Timed demos
 - ♦ Crude animation
 - ♦ Interval timers do *not* provide great precision!

Introduction to XVT-Design++ Development

Slide 6

Additional Event Details: Simulated Multitasking

❖ Remember?

- ♦ Application processes event, then gives control back to windowing system
- ♦ If applications can not process events in a timely fashion, application responsiveness degrades

❖ In certain cases, CPU- or disk-intensive operations are inevitable

❖ Two choices to provide a responsive GUI:

- ♦ Perform the operation as a state machine, using an interval timer to signal when it is time to start the next part of the operation
- ♦ Call `XVT_GlobalAPI::ProcessEvents()` to pass control to the window system to process any queued events during the operation

Additional Event Details: Event Masking

❖ All windows and dialogs have an associated event mask

- ♦ A 32-bit set of flags, one per event type

❖ Each event (eg CHAR) has a corresponding mask constant (eg EM_CHAR)

- ♦ Special event mask values EM_NONE (no events) and EM_ALL (all events) are also provided

❖ An event handler member function is called for a GUI object only if its bit is turned on within the current mask

Additional Event Details: Event Masking (cont.)

- ❖ Event masks are set to EM_ALL by `Init()`
- ❖ Use `GetEventMask()` to retrieve the current mask
- ❖ Use `SetEventMask()` to change the current mask
- ❖ An event mask may be constructed by OR'ing the event constants (eg, EM_CHAR | EM_SIZE)

Text Edit Objects

- ❖ Text edit objects are a special type of control
 - ♦ Multi-line, single-font controls for text editing
 - ♦ Different from native edit field controls
- ❖ Represented by the class `XVT_TextEdit`
- ❖ Important restriction: can only be used within windows
- ❖ Text edit objects are defined just like controls via XVT-Design++, and are automatically created and destroyed by their container window

Text Edit Objects (cont.)

- ❖ **A text edit object provides a viewport into a larger virtual page of text**
 - ♦ Text is divided into paragraphs, each terminated by a carriage return
 - ♦ Each paragraph can appear as one or more lines of text
- ❖ **Viewport window operations are handled for you**
 - ♦ Tracking the origin and autoscrolling the viewport
 - ♦ Positioning of the caret
 - ♦ Keyboard input - both printable characters and function keys
 - ♦ Mouse Selections
 - ♦ Edit and Font Menu operations

 Introduction to XVT-Design++ Development

Slide 11

Text Edit Objects (cont.)

- ❖ **Text edit object attributes you can set with XVT-Design++ determine whether:**
 - ♦ The viewport scrolls automatically when the user moves the insertion point outside of the view rectangle
 - ♦ A border is drawn around the object
 - ♦ Edit menu options Cut, Copy, and Paste are supported
 - ♦ Single or multiple paragraphs
 - ♦ An input character limit exists on single paragraph objects
 - ♦ The text can be edited or is read-only
 - ♦ Text is automatically word-wrapped at a margin
 - ♦ Insert or overwrite mode

 Introduction to XVT-Design++ Development

Slide 12

Text Edit Objects (cont.)

- ❖ **Programmatically, an application can:**
 - ♦ Insert and manipulate text
 - ♦ Retrieve any text by paragraph, line and character position
 - ♦ Set text, border and background colors
 - ♦ Set the font used to display the text
 - ♦ Modify any of the attributes
- ❖ **See the `XVT_TextEdit` member functions for more details**

Providing Clipboard Access

- ❖ **Allows information to be passed between applications**
- ❖ **The `XVT_ClipBoard` class represents the system clipboard**
 - ♦ Really a module - only one instance of the `XVT_ClipBoard` class exists
 - ♦ `Open()` and `Close()` member functions limit access to shared system resource
- ❖ **Default Edit Menu options provide user access to the clipboard**
 - ♦ Enable Cut and Copy if the user has selected something that can be put onto the clipboard
 - ♦ Enable Paste if the clipboard contains data that can be inserted

Providing Clipboard Access (cont.)

- ❖ The clipboard can hold one thing at a time, in a variety of formats
- ❖ Clipboard formats
 - ♦ CB_TEXT - ASCII text
 - ♦ CB_PICT - encapsulated XVT-Design++ Picture data
 - ♦ CB_APPL - application-specific format (arbitrary)
- ❖ Put data into the clipboard in as many formats as makes sense
- ❖ Take data off the clipboard in the most structured format your application understands

Introduction to XVT-Design++ Development

Slide 15

Printing Concepts

- ❖ Printing consists of drawing into a window derived from the XVT_PrintWin class
 - ♦ XVT_PrintWin class has an XVT_BaseDrawProto *DrawProtocol, just like XVT_DrawableContainer
 - ♦ Window object has no physical appearance
 - ♦ Print windows have a different callback protocol than screen windows
 - ♦ Virtual member functions AnotherPage() and DrawAction() must be implemented
- ❖ Page Setup... dialogs are provided when supported by the native platforms

Introduction to XVT-Design++ Development

Slide 16

Printing Protocol

- ❖ Use the new operator to create a print window
- ❖ Call the `Init()` member function to begin printing
- ❖ Initialize context (scaling, page number, etc.) in `DrawInit()`
 - ♦ Printer metrics (height, width, resolution) are available in attributes via `XVT_GlobalAPI::GetAttrValue()`
- ❖ Indicate if there is another page to print when a callback is received with `AnotherPage()`
- ❖ Draw page contents in `DrawAction()`
- ❖ The print window will automatically delete itself when printing is done
 - ♦ `AnotherPage()` returned `FALSE`

Introduction to XVT-Design++ Development

Slide 17

Page Setup

- ❖ Each `XVT_PrintWin` maintains a print record
 - ♦ Defines the destination printer and page setup
 - ♦ Internals of the data structure are platform-specific
- ❖ Allow the user to configure a print record
 - ♦ Call `XVT_GlobalAPI::PageSetup()` when the Page Setup... menu item is selected
- ❖ Store print records in files with document contents
 - ♦ `GetPrintRcd()` gets pointer to opaque data
 - ♦ `GetPrintRcdSize()` gives size of the print record
 - ♦ `SetPrintRcd()` attaches a print record to a print window
 - ♦ `ValidatePrintRcd()` confirms that a print record is still valid

Introduction to XVT-Design++ Development

Slide 18

File Handling

- ❖ **XVT++ provides a portable interface for:**
 - ♦ Obtaining a file name from the user
 - ♦ Specifying directories and file types in an abstract, portable manner
 - ♦ Handling related operations
- ❖ **Use the classes XVT_FileSpec and XVT_Directory**
 - ♦ Constructors provide ways of turning non-portable string representations into portable object representations
- ❖ **XVT_GlobalAPI functions provide portable methods for initializing these objects, and for maneuvering within the local directory structure**

Introduction to XVT-Design++ Development

Slide 19

File Handling (cont.)

- ❖ **XVT_GlobalAPI member functions for obtaining a file name from the user in an XVT_FileSpec object**
 - ♦ `FL_STATUS XVT_GlobalAPI::OpenFile(...)`
 - ♦ `FL_STATUS XVT_GlobalAPI::SaveFile(...)`
 - ♦ Code is automatically generated by setting a connection with XVT-Design++
- ❖ **XVT_GlobalAPI provides methods for creating XVT_Directory objects portably**
 - ♦ `XVT_GlobalAPI::GetDefaultDir()`
 - ♦ `XVT_GlobalAPI::GetDir()`

Introduction to XVT-Design++ Development

Slide 20

Error Managers

- ❖ Chains of error handlers are maintained by the error manager for a specific type of error
 - ♦ Subclasses of `XVT_ErrorManager` can be instantiated to manage a specific type of error
 - ♦ XVT++ provides two classes: `XVT_InternalErrorManager` and `XVT_AllocErrorManager`
- ❖ Use the `Raise()` member function of the error manager class to signal an error condition to the first error handler in the chain
- ❖ Error handler return values indicate whether the error has been handled or not

 Introduction to XVT-Design++ Development

Slide 21

Error Handlers

- ❖ Derive classes from `XVT_ErrorHandler` to handle specific types of errors
 - ♦ The constructor must provide the error manager the type of error handler it belongs to
 - ♦ You must provide an implementation of the virtual function `Handle()`
 - ♦ XVT++ provides two classes : `XVT_InternalErrorHandler` and `XVT_AllocErrorHandler` - derive from these or derive from the base class to handle other types of errors
- ❖ `Handle()` returns `TRUE` if the error condition is resolved, or `FALSE` if the error manager should pass the error to the next handler in the chain
- ❖ The error manager maintains chains of error handlers for each type of error

 Introduction to XVT-Design++ Development

Slide 22

Handling User Errors

- ❖ **Plan for user errors**
 - ♦ Make them as low impact as possible
 - ♦ Allow the user to recover
- ❖ **For handling recoverable errors, set connections in XVT-Design++ to**
 - ♦ XVT_NOTE - display an informational message
 - ♦ XVT_ERROR - display an error message
 - ♦ XVT_ASK - ask the user a question
- ❖ **XVT_GlobalAPI::Beep() can be called to produce an error tone**

Handling Fatal Errors

- ❖ **When your program detects an unrecoverable error, it can put up a dialog box and terminate execution with a call to**
`XVT_GlobalAPI::Fatal()`
- ❖ **To put up this dialog without terminating you can call**
`XVT_GlobalAPI::Message()`
- ❖ **XVT++ reports its internal errors to you via**
`XVT_InternalErrorManager::Raise()`
 - ♦ Usually reported for illegal usage of API functions
 - ♦ Internal error numbers and descriptions are written to the file XVTPEERR.TXT

Handling Memory Allocation Errors

- ❖ An error is raised by `XVT_AllocErrorManager` if a `new` or `malloc()` operation fails
- ❖ Subclass `XVT_AllocErrorHandler` to implement an error handling policy
 - ♦ Implement the pure virtual member function `Handler()`
 - ♦ Returning `TRUE` indicates that the allocation should be tried again
 - ♦ Useful for implementing “rainy day memory fund” algorithms
 - ♦ Internally the XVT++ class libraries use both `malloc()` and `new` - you may need to release space on both heaps

Introduction to XVT-Design++ Development

Slide 25

Debugging and Tracing

- ❖ For tracing, use `XVT_GlobalAPI::Debug()` or `XVT_GlobalAPI::Debug2()`
 - ♦ `Debug` writes trace info to a file named `DEBUG`
 - ♦ `Debug2` implements `Debug` with compile/run-time on/off switches
- ❖ `Printf`-style output will work on X-based platforms
- ❖ Using `Note()` or `Message()` will change the event flow of your program

Introduction to XVT-Design++ Development

Slide 26

Memory Management

- ❖ **XVT++ provides a complete encapsulation layer for memory management of the “global” heap**
 - ♦ Global heap memory represented by a handle (GHANDLE) which must be locked to access a pointer to the memory
 - ♦ GHANDLEs allow the OS memory manager to rearrange blocks of global memory
 - ♦ Global heap exists only on the Macintosh and MS-Windows
- ❖ **XVT_GlobalAPI provides member functions for**
 - ♦ Global heap memory allocation and deallocation
 - ♦ Locking and unlocking global heap memory

 Introduction to XVT-Design++ Development

Slide 27

Help System

- ❖ **A topic-oriented, two-level help message browser (Example: XVT-Design++ Help facility)**
 - ♦ Topics dialog
 - ♦ Help message dialog
 - ♦ Not context-sensitive
- ❖ **To display the Help Topics dialog, call**
`XVT_GlobalAPI::Help()`
- ❖ **The help system GUI is itself written in XVT**
 - ♦ The Help source file is vhelp.c

 Introduction to XVT-Design++ Development

Slide 28

Help System (cont.)

- ❖ **To supply an application with a help text file:**
 - ♦ Type the help text into a file with a text editor
 - ♦ Compile this file with the CCHelp program
 - ♦ Ship the compiled help file with your application
 - ♦ Instruct the user to install it along with the application
- ❖ **An application finds its help text in a file at run time when help is requested by the user**

Help System (cont.)

- ❖ **The general format for a help file is:**

```
@topic-one
Text line one for topic one
Text line two for topic one
...
* Comment line
@topic-two
Text line one for topic two
Text line two for topic two
...
```

Help System (cont.)

❖ **Formatting guidelines:**

- ♦ The order of the topics in the help source file will be the order in the topics dialog box
- ♦ Indent a topic under another in the topics dialog box to indicate a subtopic (put a tab between the @ and the name of the subtopic)
- ♦ No word-wrap is performed and different platforms may use fixed width or proportional fonts

❖ **Compile your help source with CCHelp**

- ♦ A conventional command-line program
- ♦ Produces a binary compressed form of the original ASCII help file
- ♦ You specify input and output file names

System Attribute Table

❖ **The System Attribute Table (SAT) maintains a collection of attribute names and values**

❖ **Portable attributes, such as screen resolution and dimensions, control sizes, and system configuration parameters are available, as well as platform-specific attributes**

- ♦ A complete list of portable attributes can be found in the *XVT-Design++ 2.0 Class Library Reference*
- ♦ Non-portable attributes are documented in the Platform-Specific Books

System Attribute Table (cont.)

- ❖ The SAT is accessed by two XVT_GlobalAPI member functions: `GetAttrValue()` and `SetAttrValue()`

```
long GetAttrValue (XVT_Base* win, long attribute);  
void SetAttrValue (XVT_Base* win, long attribute, long value);
```

- ❖ Some attributes are read-only (cannot be set)
- ❖ Use the NULL argument for those attributes not associated with a specific GUI object
- ❖ All attributes begin with the prefix `ATTR_`
- ❖ Non-portable attributes begin with a platform-specific prefix (such as `ATTR_WIN_`)

System Attribute Table (cont.)

- ❖ Example of using a portable attribute:

```
XVY_GlobalAPI *gapi;  
dx += (int)gapi->GetAttrValue(NULL,ATTR_DOC_STAGGER_HORZ);
```

- ❖ Example of using a non-portable attribute:

```
#if (XVTWS == XOLWS) || (XVTWS == MTFWS)  
XVY_GlobalAPI *gapi;  
gapi->SetAttrValue( NULL, ATTR_X_MASK_SERVER_EVENTS, (long)TRUE );  
#endif
```


Writing Non-Portable Code

- ❖ Use `#ifdefs` to protect any platform-specific code you write
- ❖ Preprocessor constants defined by `xvtp.h` identify
 - ♦ Native Window System (ie `#if XVTWS == MTFWS`)
 - ♦ Operating System (ie `#if XVT_OS == XVT_OS_AIX`)
 - ♦ Compiler (ie `#if XVT_CC == XVT_CC_MPW`)
- ❖ General guidelines:
 - ♦ Build a portable interface to non-portable code
 - ♦ Use separate modules for non-portable code
 - ♦ Avoid using non-portable functionality that is critical to the application user interface - it may be a large amount of work to emulate it on other platforms

Non-Portable Code: Hook Functions

- ❖ Supply a hook function to modify keystroke processing or intercept native events
- ❖ Access the native event information before XVT++ processes the events
- ❖ The hook functions are set at run-time by using:

```
SetAttrValue(NULL, ATTR_KEY_HOOK, (long)<key_hook>);  
SetAttrValue(NULL, ATTR_EVENT_HOOK, (long)<event_hook>);
```

where `<event_hook>` and `<key_hook>` are replaced by the name of the hook function supplied by the application

Non-Portable Code: Hook Functions (cont.)

- ❖ Hook functions are completely non-portable, since they will receive platform-specific events
- ❖ If the supplied hook function returns FALSE, XVT++'s default function will be called as well

```
int HookFunction(Widget widget, int event, void *data) {
    if (event == XVT_HOOK_EVENT_MOUSE_CLICK) {
        // Handle mouse click event
        return TRUE;
    }
    // If event is not handled, return FALSE so default function is called
    return FALSE;
}
```

Introduction to XVT-Design++ Development

Slide 37

Non-Portable Code: Native Toolkit Access

- ❖ XVT++'s layered API approach allows access to the native toolkit
- ❖ You need to understand the effects of native function calls on the Design++ library
 - ♦ Read the XVT-Design++ appropriate Platform-Specific Book
 - ♦ Relative expertise in native GUI programming tools required
 - ♦ Look forward to experimentation and testing
 - ♦ Remember: you may need to find analogous functionality on other target platforms

Introduction to XVT-Design++ Development

Slide 38

Building the Final Application

- ❖ Generate a fresh version of your application via XVT-Design++
- ❖ Build and link the application
- ❖ Use CURL to convert URL resources to native resources, then compile the native resources
- ❖ make the application
- ❖ Test and refine as required

Introduction to XVT-Design++ Development

Slide 39

Course Example -Summary

- ❖ Offer different strategies for using XVT C++ tools (and your existing development tools) to their best combined advantage
 - ♦ Code from within the XVT-Design++ ACE
 - ♦ Code outside of XVT-Design++
- ❖ Build more familiarity with XVT-Design++ and XVT++'s Class Library and API
- ❖ Suggestions for XVT-Design++ application structuring

Introduction to XVT-Design++ Development

Slide 40

Course Example - Summary (cont.)

❖ Examples of specific XVT-Design++ programming techniques:

- ♦ Planning and designing an application
- ♦ Project files and system attributes
- ♦ Using externally-defined windows, dialogs, controls, menus, and strings
- ♦ Managing windows and dialogs (and their contained controls)
- ♦ Window, dialog, and control attributes
- ♦ Prototyping with connections and TestMode
- ♦ Using pre-defined dialogs
- ♦ Managing events and utilizing special tags
- ♦ Graphics drawing, drawing tools, and graphics transformations

Introduction to XVT-Design++ Development

Slide 41

Additional Sources of Assistance

❖ Customer Support

- ♦ Phone 303-443-8988 between 8 AM and 5 PM Monday-Friday
- ♦ E-mail via Internet at techsup@xvt.com
- ♦ E-mail via CompuServe at 75765,1233
- ♦ E-mail via XVT BBS at 303-443-9083 (2400 baud) or 303-443-7780 (9600 baud)
- ♦ Write to XVT Software Inc., P O Box 18750, Boulder CO 80308

❖ xvtdev Mailing List

❖ Annual XVT Developer's Conference

❖ XVT's Consulting and Training Group (CTG)

- ♦ Offers extensive fee-based services
- ♦ Includes training, consulting, ports, etc.
- ♦ Phone 303-443-4223 for more information

Introduction to XVT-Design++ Development

Slide 42

Commencement

❖ Good luck with your XVT-Design++ application development

❖ Let us know if we can be of further assistance

Introduction to XVT-Design++ Development

Slide 43

NOTES:

Communication

Good link with your team

to ensure

to ensure it is clear

to ensure it is clear

